web.py

目錄

Web.py Cookbook 简体中文版

> 来源：Web.py Cookbook 简体中文版

欢迎来到web.py 0.3的Cookbook。提醒您注意：某些特性在之前的版本中并不可用。当前开发版本是0.3。

格式

1. 在编排内容时，请尽量使用cookbook格式...如：

    ### 问题：如何访问数据库中的数据？

    ### 解法：使用如下代码...

2. 请注意，网址中不必含有"web"。如"/cookbook/select"，而非"/cookbook/web.select"。
3. 该手册适用于0.3版本，所以您在添加代码时，请确认代码能在新版本中工作。

基本应用:

高级应用

Sessions and user state 会话和用户状态:

- 如何使用Session
- 如何在调试模式下使用Session
- 在template中使用session
- 如何操作Cookie
- 用户认证
- 一个在postgreSQL数据库环境下的用户认证的例子
- 如何在子应用中操作Session

Utils 实用工具:

- 如何发送邮件
- 如何利用Gmail发送邮件
- 使用soaplib实现webservice

Templates 模板

- Templetor: web.py 模板系统
- 使用站点布局模板
- 交替式风格 (未译)
- 导入函数到模板中 (未译)
- 模板文件中的i18n支持
- 在web.py中使用Mako模板引擎
- 在web.py中使用Cheetah模板引擎
- 在web.py中使用Jinja2模板引擎
- 如何在谷歌应用程序引擎使用模板

Testing 测试:

- Testing with Paste and Nose (未译)
- RESTful doctesting using an application's request method (未译)

User input 用户输入:

- 文件上传
- 保存上传的文件
- 上传文件大小限定
- 通过 web.input 接受用户输入
- 怎样使用表单
- 显示个别表单字段

Database 数据库

- 使用多数据库
- Select: 查询数据
- Update: 更新数据
- Delete: 删除数据
- Insert: 新增数据
- Query: 高级数据库查询
- 怎样使用数据库事务
- 使用 sqlalchemy
- 整合 SQLite UDF (用户定义函数) 到 webpy 数据库层
- 使用字典动态构造where子句

Deployment 部署:

- 通过Fastcgi和lighttpd部署
- 通过Webpy和Nginx with FastCGI搭建Web.py
- CGI deployment through Apache (未译)
- mod_python deployment through Apache (requested)
- 通过Apache和mod_wsgi部署
- mod_wsgi deployment through Nginx (未译)
- Fastcgi deployment through Nginx (未译)

Subdomains 子域名:

- Subdomains and how to access the username (requested)

安装

Summary

安装

安装web.py, 请先下载：

```
http://webpy.org/static/web.py-0.37.tar.gz
```

或者获取最新的开发版：

```
https://github.com/webpy/webpy/tarball/master
```

解压并拷贝 *web* 文件夹到你的应用程序目录下。 或者，为了让所有的应用程序都可以使用，运行：

```
python setup.py install
```

注意: 在某些类unix系统上你可能需要切换到root用户或者运行：

```
sudo python setup.py install
```

查看 推荐设置.

另外一个选择是使用Easy Install. Easy Install 使用如下：

```
easy_install web.py
```

或者 PIP

```
sudo pip install web.py
```

开发

web.py 内置了web服务器。可以按照 tutorial 学习如何写一个Web应用。 写完后，将你的代码放到 `code.py` 并如下面的方法来启动服务器：

```
 python code.py
```

打开你的浏览器输入 http://localhost:8080/ 查看页面。 若要制定另外的端口，使用 `python code.py 1234` 。

产品

现在所运行 web.py 程序的web服务器是挺不错的， 但绝大多数网站还是需要更加专业一些的web服务器。web.py 实现了 WSGI 并能在任何兼容它的服务器上运行。 WSGI 是一个web服务器与应用程序之间的通用API, 就如Java 的 Servlet 接口。 你需要安装 flup (download here) 使web.py 支持with CGI， FastCGI 或 SCGI， flup提供了这些API的WSGI接口。

对于所有的CGI变量， 添加以下到你的 `code.py` :

```
#!/usr/bin/env python
```

并运行 `chmod +x code.py` 添加可执行属性。

LightTPD

.. 使用 FastCGI

在产品中通过FastCGI结合lighttpd是web.py使用的一种推荐方法。 reddit.com 通过该方法来处理百万次的点击。

lighttpd config设置参考如下：

```
server.modules = ("mod_fastcgi", "mod_rewrite")
server.document-root = "/path/to/root/"
fastcgi.server = ( "/code.py" =>
(( "socket" => "/tmp/fastcgi.socket",
   "bin-path" => "/path/to/root/code.py",
   "max-procs" => 1
))
)

url.rewrite-once = (
  "^/favicon.ico$" => "/static/favicon.ico",
  "^/static/(.*)$" => "/static/$1",
  "^/(.*)$" => "/code.py/$1"
)
```

在某些版本的lighttpd中， 需要保证fastcgi.server选项下的"check-local"属性设置为"false", 特别是当你的 `code.py` 不在文档根目录下。

如果你得到错误显示不能够导入flup， 请在命令行下输入 "easy_install flup" 来安装。

从修订版本 145开始， 如果你的代码使用了重定向，还需要在fastcgi选项下设置bin-environment变量。 如果你的代码重定向到http://domain.com/ 而在url栏中你会看到 http://domain.com/code.py/， 你可以通过设置这个环境变量来阻止。 这样你的fastcgi.server设置将会如下：

```
fastcgi.server = ( "/code.py" =>
((
   "socket" => "/tmp/fastcgi.socket",
   "bin-path" => "/path/to/root/code.py",
   "max-procs" => 1,
   "bin-environment" => (
     "REAL_SCRIPT_NAME" => ""
   ),
   "check-local" => "disable"
))
)
```

Apache

.. 使用 CGI

添加以下到 `httpd.conf` 或 `apache2.conf` 。

```
Alias /foo/static/ /path/to/static
ScriptAlias /foo/ /path/to/code.py
```

.. 使用 CGI .htaccess

CGI很容易配置，但不适合高性能网站。添加以下到你的 `.htaccess` ：

```
Options +ExecCGI
AddHandler cgi-script .py
```

将你的浏览器指向 `http://example.com/code.py/` 。不要忘记最后的斜杠，否则你将会看到 `not found` 消息(因为在 `urls` 列表中你输入的没有被匹配到). 为了让其运行的时候不需要添加 `code.py` ，启用mod_rewrite 法则 (查看如下)。

注意: `web.py` 的实现破坏了 `cgitb` 模块，因为它截取了 `stdout` 。可以通过以下的方法来解决该问题：

```
import cgitb; cgitb.enable()
import sys

# ... import web etc here...

def cgidebugerror():
    """
    """        _wrappedstdout = sys.stdout

    sys.stdout = web._oldstdout
    cgitb.handler()

    sys.stdout = _wrappedstdout

web.internalerror = cgidebugerror
```

.. 使用 FastCGI

FastCGI很容易配置，运行方式如同mod_python。

添加以下到 `.htaccess` ：

```
<Files code.py>      SetHandler fastcgi-script
</Files>
```

不幸的是, 不像lighttpd, Apache不能够暗示你的web.py脚本以FastCGI 服务器的形式工作，因此你需要明确的告诉web.py。添加以下到 `code.py` 的 `if __name__ == "__main__":` 行前：

```
web.wsgi.runwsgi = lambda func, addr=None: web.wsgi.runfcgi(func, a
```

将你的浏览器指向 `http://example.com/code.py/` 。不要忘记最后的斜杠，否则你将会看到 `not found` 消息(因为在 `urls` 列表中你输入的没有被匹配到).为了让其运行的时候不需要添加 `code.py` ，启用mod_rewrite 法则 (查看如下)。

Walter 还有一些额外建议.

.. 使用 SCGI

https://www.mems-exchange.org/software/scgi/ 从 http://www.mems-exchange.org/software/files/mod_scgi/ 下载 `mod_scgi` 代码 windows apache 用户： 编辑 httpd.conf：

```
LoadModule scgi_module Modules/mod_scgi.so
SCGIMount / 127.0.0.1:8080
```

重启apache，并在命令行中如下方式启动code.py：

```
python code.py 127.0.0.1:8080 scgi
```

打开你的浏览器，访问127.0.0.1 It's ok!

.. 使用 mod_python

mod_python 运行方式如同FastCGI， 但不是那么方便配置。

对于 Python 2.5 按照如下：

```
cd /usr/lib/python2.5/wsgiref
# or in windows: cd /python2.5/lib/wsgiref
wget -O modpython_gateway.py http://projects.amor.org/misc/browser/
# or fetch the file from that address using your browser
```

对于 Python <2.5 按照如下：

```
cd /usr/lib/python2.4/site-packages
# or in windows: cd /python2.4/lib/site-packages
svn co svn://svn.eby-sarna.com/svnroot/wsgiref/wsgiref
cd wsgiref
wget -O modpython_gateway.py http://projects.amor.org/misc/browser/
# or fetch the file from that address using your browser
```

重命名 `code.py` 为 `codep.py` 或别的名字，添加：

```
app = web.application(urls, globals())
main = app.wsgifunc()
```

在 `.htaccess` 中，添加：

```
AddHandler python-program .py
PythonHandler wsgiref.modpython_gateway::handler
PythonOption wsgi.application codep::main
```

你应该希望添加 `RewriteRule` 将 `/` 指向 `/codep.py/`

确保访问 `/codep.py/` 的时候有添加最后的 `/` 。否则，你将会看到一条错误信息，比如 `A server error occurred. Please contact the administrator.`

.. 使用 mod_wsgi

mod_wsgi 是一个新的Apache模块 通常优于mod_python 用于架设WSGI应用，它非常容易配置。

在 `code.py` 的最后添加：

```
app = web.application(urls, globals(), autoreload=False)
application = app.wsgifunc()
```

mod_wsgi 提供 许多可行方法 来实现WSGI应用, 但一种简单的方法是添加以下到.htaccess：

```
<Files code.py>
    SetHandler wsgi-script
    Options ExecCGI FollowSymLinks
</Files>
```

如果在apache的 error.log 文件中出现 "ImportError: No module named web"， 在导入web之前，你可能需要在code.py中尝试设置绝对路径：

```
import sys, os
abspath = os.path.dirname(__file__)
sys.path.append(abspath)
os.chdir(abspath)
import web
```

同时， 你可能需要查看 WSGI应用的常见问题的 "Application Working Directory" 部分。

最终应该可以访问 `http://example.com/code.py/` 。

mod_rewrite 法则，Apache

如果希望 web.py 能够通过 'http://example.com' 访问，代替使用 'http://example.com/code.py/'， 添加以下法则到 `.htaccess` 文件：

```
<IfModule mod_rewrite.c>
  RewriteEngine on
  RewriteBase /
  RewriteCond %{REQUEST_URI} !^/icons
  RewriteCond %{REQUEST_URI} !^/favicon.ico$
  RewriteCond %{REQUEST_URI} !^(/.*)+code.py/
  RewriteRule ^(.*)$ code.py/$1 [PT]
</IfModule>
```

如果 `code.py` 在子目录 `myapp/` 中， 调整 RewriteBase 为 `RewriteBase /myapp/` 。 如果还有一些静态文件如CSS文件和图片文件, 复制这些并改成你需要的地址。

web.py 0.3 新手指南

开始

你知道Python同时你希望制作一个网站。 那么web.py正好提供了一种简单的方法。

如果你希望读完整个指南， 你需要安装Python, web.py, flup, psycopg2, 和Postgres (或者等价的数据库和Python驱动)。 详细，可以查看 webpy.org.

如果你已经有了一个web.py项目，请看看升级 页面的相关信息。

准备开始。

URL 处理

任何网站最重要的部分就是它的URL结构。你的URL并不仅仅只是访问者所能看到并且能发给朋友的。它还规定了你网站运行的心智模型。在一些类似del.icio.us的流行网站 , URL甚至是UI的一部分。 web.py使这类强大的URL成为可能。

在开始你的web.py程序之前,打开一个文本文件（文件名为code.py）输入:

```
import web
```

这条语句会导入web.py模块。

现在我们需要把我们的URL结构告诉web.py。让我从下面这个简单的例子开始:

```
urls = (
  '/', 'index'
)
```

第一部分是匹配URL的正则表达式，像 `/` 、 `/help/faq` 、 `/item/(\d+)` 等 (`\d+` 将匹配数字)。圆括号表示捕捉对应的数据以便后面使用。第二部分是接受请求的类名称，像 `index` 、 `view` 、 `welcomes.hello` (`welcomes` 模块的 `hello` 类)，或者 `get_\1` 。 `\1` 会被正则表达式捕捉到的内容替换，剩下来捕捉的的内容将被传递到你的函数中去。

这行表示我们要URL `/` (首页)被一个叫 `index` 的类处理。

现在我们需要创建一个列举这些url的application。

```
app = web.application(urls, globals())
```

这会告诉web.py去创建一个基于我们刚提交的URL列表的application。这个application会在这个文件的全局命名空间中查找对应类。

GET和POST: 区别

现在我们需要来写 `index` 类。虽然大多数人只会看看，并不会注意你的浏览器在使用用于与万维网通信的HTTP语言。具体的细节并不重要，但是要理解web访问者请求web服务器去根据URL(像 `/` 、 `/foo?f=1`)执行一个合适的函数（像 `GET` 、 `POST` ）的基本思想。

`GET` 是我们都熟悉的。它用于请求网页文本。当你在浏览器输入 `harvard.edu` ，它会直接访问Harvard的web服务器，去 `GET /` 。 第二个最有名的是 `POST` ，它经常被用在提交form，比如请求买什么东西。每当提交一个去做什么事情(像使用信用卡处理一笔交易)的请求时，你可以使用 `POST` 。这是关键，因为 `GET` 的URL可以被搜索引擎索引，并通过搜索引擎访问。虽然大部分页面你希望被索引，但是少数类似订单处理的页面你是不希望被索引的 (想象一下Google尝试去购买你网站上的所有东西)。

在我们web.py的代码中，我们将这两个方法明确区分:

```
class index:
    def GET(self):
        return "Hello, world!"
```

当有人用 `GET` 请求 `/` 时，这个 `GET` 函数随时会被web.py调用。

好了，限制我们只需要最后一句就写完了。这行会告诉web.py开始提供web页面:

```
if __name__ == "__main__": app.run()
```

这会告诉web.py为我们启动上面我们写的应用。

现在注意，即使我已经在这里说了很多，但我们真正有5行这些代码。这就是你需要编写的一个完整的web.py应用。 为了更方便的使用，你的完整代码应该像下面这样:

```
import web

urls = (
    '/', 'index'
)

class index:
    def GET(self):
        return "Hello, world!"

if __name__ == "__main__":
    app = web.application(urls, globals())
    app.run()
```

启动服务

如果你在命令行下面，请输入:: $ python code.py http://0.0.0.0:8080/

现在你的web.py应用正运行在你电脑上的一个真正的web服务器上。 访问那个URL，然后你应该看到"Hello, world!" (你可以通过把IP地址/端口加在"code.py"的后面，来控制web.py在哪里启动服务器。你也可以让它运行在 `fastcgi` 或 `scgi` 服务器上)。

注意: 如果你不能或者不想使用默认端口，你可以使用这样的命令来指定端口号:

```
$ python code.py 1234
```

模板

在 Python 中写 HTML 不是聪明的选择，相反在 HTML 中写 Python 则有趣的多。幸运的是， `web.py` 让这件事情做得简单而又漂亮。

注意： 老版本的 `web.py` 使用 Cheetah 模板系统，你可以也欢迎使用其他模板系统，但它可能不会被长久支持。

给模板新建一个目录（命名为 `templates` ），在该目录下新建一个以 `.html` 结尾的文件，内容如下：

```
<em>Hello</em>, world!
```

你也可以在模板中使用 `web.py` 模板支持代码：

```
$def with (name)

$if name:
    I just wanted to say <em>hello</em> to $name.
$else:
    <em>Hello</em>, world!
```

如上，该模板看起来就像 python 文件一样，除了顶部的 `def with` (表示从模板将从这后面取值)和总是位于代码段之前的 `$` 。当前， `template.py` 首先请求模板文件的首行 `$def` 。当然，你要注意 `web.py` 将会转义任何任何用到的变量，所以当你将 `name` 的值设为是一段 HTML 时，它会被转义显示成纯文本。如果要关闭该选项，可以写成 `$:name` 来代替 `$name` 。

回看再看 `code.py` 。在第一行之下添加：

```
render = web.template.render('templates/')
```

这会告诉web.py到你的模板目录中去查找模板。然后把 `index.GET` 改成: 告诉 `web.py` 在你的模板目录下查找模板文件。修改 `index.GET` ：

```
name = 'Bob'
return render.index(name)
```

（'index' 是模板的名字，'name' 是传入模板的一个参数）

访问站点它将显示 hello Bob。

但是如果我们想让用户自行输入他的名字，么办？如下：

```
i = web.input(name=None)
return render.index(i.name)
```

访问 `/` 将显示 hello world，访问 `/?name=Joe` 将显示 hello Joe。

URL 的后面的 `?` 看起来不好看？修改下 URL 配置：

```
'/(.*)', 'index'
```

然后修改下 `index.GET` ：

```
def GET(self, name):
    return render.index(name)
```

现在访问 `/Joe` 看看，它会显示 hello Joe。

如果学习更多关于 web.py 的模板处理，请访问 web.py 模板.

表单

web.py的form模块能够帮助你生成HTML表单；获取用户的输入，并在处理或添加到数据库之前对其进行内容的验证。 如果你要学习更多关于form模块的使用，请查看帮助文档或者Form类库的链接

数据库操作

注意: 在你开始使用数据库之前，确保你已经安装了合适的数据库访问库。比如对于MySQL数据库，使用 MySQLdb ，对于Postgres数据库使用psycopg2。

首先你需要创建一个数据库对象。

```
db = web.database(dbn='postgres', user='username', pw='password', d
```

(根据需要修改这里 -- 尤其是 `username` 、 `password` 、 `dbname` -- 。 MySQL用户还需要把 `dbn` 定义改为 `mysql` 。)

这就是所有你需要做的 -- web.py将会自动处理与数据库的连接和断开。

使用的的数据库引擎管理工具，在你的库中创建一个简单的表:

```
CREATE TABLE todo (
  id serial primary key,
  title text,
  created timestamp default now(),
  done boolean default 'f'    );
```

然后初始化行:

```
INSERT INTO todo (title) VALUES ('Learn web.py');
```

我们回来继续编辑 `code.py` ，把 `index.GET` 改成下面的样子，替换整个函数:

```
def GET(self):
    todos = db.select('todo')
    return render.index(todos)
```

然后把URL列表改回来，只保留 `/` :

```
'/', 'index',
```

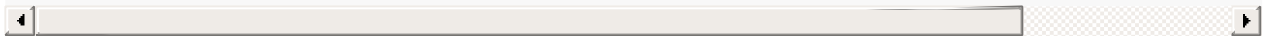
像这样编辑并替换 `index.html` 的全部内容:

```
$def with (todos)
<ul>
$for todo in todos:
    <li id="t$todo.id">$todo.title</li>
</ul>
```

再访问你的网站，然后你可以看到你的todo item: "Learn web.py"。恭喜你！你已经完整地写好了一个可以从数据库读取数据的程序。现在让我们同样再写一个可以把数据写入数据库的程序。

在 `index.html` 尾部添加:

```
<form method="post" action="add">
<p><input type="text" name="title" /> <input type="submit" value="A
</form>
```



然后把你的URL列表改为:

```
'/', 'index',
'/add', 'add'
```

(你必须要非常小心那些逗号。如果你省略他们，Python会把所有字符串连接起来, 变成 `'/index/addadd'`)

现在添加另一个类:

```
class add:
    def POST(self):
        i = web.input()
        n = db.insert('todo', title=i.title)
        raise web.seeother('/')
```

(注意现在我们正在使用 `POST`)

`web.input` 可以让你访问用户通过form提交的任何数据。

注意: 如果要访问多个相同名字的字段，请使用list的格式(比如:一串name="name"的多选框):

```
post_data=web.input(name=[])
```

`db.insert` 把数据插入数据表 `todo` ，然后把新的行号返回给你。
`seeother` 把用户重定向到指定的URL。

一些快速补充说明: `db.update` 与 `db.insert` 差不多，除了它返回的行号是直接从sql语句里面提取的(`WHERE ID=2`)。

`web.input` 、 `db.query` 已经其他web.py中的函数返回"Storage objects"，这些东西就像字典，你除了可以 `d['foo']` 之外，你还可以 `d.foo` 。这可以让代码更加干净。

你可以在the documentation找到这方面具体的细节以及所有web.py的函数说明。

开发

web.py 还有一些帮助我们debug的工具。当它在内建的服务器中运行时，它会一debug模式启动程序。在debug模式中，任何代码、模板的修改，都会让服务器重新加载它们，然后还会输出有用的错误消息。

只有在生产环境中debug模式是关闭的。如果你想禁用debug模式，你可以在创建程序/模板前添加像这样的行。

```
web.config.debug = False
```

我们的指南就到这里了。如果要做更多很酷的东西，你可以先查看一下文档。

下一步是什么?

- 更多文档
- Cookbook
- code samples

基本应用

Hello World!

问题

如何用web.py实现Hello World!？

解法

```
import web

urls = ("/.*", "hello")
app = web.application(urls, globals())

class hello:
    def GET(self):
        return 'Hello, world!'

if __name__ == "__main__":
    app.run()
```

提示：要保证网址有无'/'结尾，都能指向同一个类。就要多写几行代码，如下：

在URL开头添加代码：

```
'/(.*)/', 'redirect',
```

然后用redirect类处理以'/'结尾的网址：

```
class redirect:
    def GET(self, path):
        web.seeother('/' + path)
```

提供静态文件 (诸如js脚本, css样式表和图象文件)

问题

如何在web.py自带的web server中提供静态文件访问？

解法

web.py 服务器

在当前应用的目录下，创建一个名为static的目录，把要提供访问的静态文件放在里面即可。

例如, 网址 `http://localhost/static/logo.png` 将发送 `./static/logo.png` 给客户端。

Apache

在 Apache 中可以使用 Alias 指令，在处理 web.py 之前将请求映射到指定的目录。

这是一个在 Unix like 系统上虚拟主机配置的例子：

```
<VirtualHost *:80>
    ServerName example.com:80
    DocumentRoot /doc/root/
    # mounts your application if mod_wsgi is being used
    WSGIScriptAlias / /script/root/code.py
    # the Alias directive
    Alias /static /doc/root/static

    <Directory />
        Order Allow,Deny
        Allow From All
        Options -Indexes
    </Directory>

    # because Alias can be used to reference resources outside docu
    # must reference the directory with an absolute path
    <Directory /doc/root/static>
        # directives to effect the static directory
        Options +Indexes
    </Directory>
</VirtualHost>
```

理解URL控制

`问题`: 如何为整个网站设计一个URL控制方案 / 调度模式

`解决`:

web.py的URL控制模式是简单的、强大的、灵活的。在每个应用的最顶部，你通常会看到整个URL调度模式被定义在元组中:

```
urls = (
    "/tasks/?", "signin",
    "/tasks/list", "listing",
    "/tasks/post", "post",
    "/tasks/chgpass", "chgpass",
    "/tasks/act", "actions",
    "/tasks/logout", "logout",
    "/tasks/signup", "signup"
)
```

这些元组的格式是: *URL*路径, 处理类 这组定义有多少可以定义多少。如果你并不知道URL路径和处理类之间的关系，请在阅读cookbook之前先阅读Hello World example，或者快速入门。

`路径匹配`

你可以利用强大的正则表达式去设计更灵活的URL路径。比如 /(test1|test2) 可以捕捉 /test1 或 /test2。要理解这里的关键，匹配是依据URL路径的。比如下面的URL:

```
http://localhost/myapp/greetings/hello?name=Joe
```

这个URL的路径是 */myapp/greetings/hello*。web.py会在内部给URL路径加上^ 和$ ，这样 */tasks/* 不会匹配 */tasks/addnew*。URL匹配依赖于“路径”，所以不能这样使用，如： */tasks/delete?name=(.+)* ,?之后部分表示是“查询”，并不会被匹配。阅读URL组件的更多细节，请访问web.ctx。

`捕捉参数`

你可以捕捉URL的参数，然后用在处理类中:

```
/users/list/(.+), "list_users"
```

在 *list/*后面的这块会被捕捉，然后作为参数被用在GET或POST:

```
class list_users:
    def GET(self, name):
        return "Listing info about user: {0}".format(name)
```

你可以根据需要定义更多参数。同时要注意URL查询的参数(?后面的内容)也可以用web.input()取得。

`开发子程序的时候注意`

为了更好的控制大型web应用，web.py支持子程序。在为子程序设计URL模式的时候，记住取到的路径(web.ctx.path)是父应用剥离后的。比如，你在主程序定义了URL"/blog"跳转到'blog'子程序，那没在你blog子程序中所有URL都是以"/"开头的，而不是"/blog"。查看web.ctx取得更多信息。

使用子应用

问题

如何在当前应用中包含定义在其他文件中的某个应用？

解法

在 `blog.py` 中:

```
import web
urls = (
  "", "reblog",
  "/(.*)", "blog"
)

class reblog:
    def GET(self): raise web.seeother('/')

class blog:
    def GET(self, path):
        return "blog " + path

app_blog = web.application(urls, locals())
```

当前的主应用 `code.py`:

```
import web
import blog
urls = (
  "/blog", blog.app_blog,
  "/(.*)", "index"
)

class index:
    def GET(self, path):
        return "hello " + path

app = web.application(urls, locals())

if __name__ == "__main__":
    app.run()
```

提供XML访问

问题

如何在web.py中提供XML访问？

如果需要为第三方应用收发数据，那么提供xml访问是很有必要的。

解法

根据要访问的xml文件(如response.xml)创建一个XML模板。如果XML中有变量，就使用相应的模板标签进行替换。下面是一个例子：

```
$def with (code)
<?xml version="1.0"?>
<RequestNotification-Response>
<Status>$code</Status>
</RequestNotification-Response>
```

为了提供这个XML，需要创建一个单独的web.py程序(如response.py)，它要包含下面的代码。注意：要用"web.header('Content-Type', 'text/xml')"来告知客户端——正在发送的是一个XML文件。

```
import web

render = web.template.render('templates/', cache=False)

urls = (
    '/(.*)', 'index'
)

app = web.application(urls, globals())

class index:
    def GET(self, code):
        web.header('Content-Type', 'text/xml')
        return render.response(code)

web.webapi.internalerror = web.debugerror
if __name__ == '__main__': app.run()
```

从post读取原始数据

介绍

有时候，浏览器会通过post发送很多数据。在webpy，你可以这样操作。

代码

```
class RequestHandler():
    def POST():
        data = web.data() # 通过这个方法可以取到数据
```

高级应用

web.ctx

问题

如何在代码中得到客户端信息？比如：来源页面(referring page)或是客户端浏览器类型

解法

使用web.ctx即可。首先讲一点架构的东西：web.ctx基于threadeddict类，又被叫做ThreadDict。这个类创建了一个类似字典(dictionary-like)的对象，对象中的值都是与线程id相对应的。这样做很妙,因为很多用户同时访问系统时，这个字典对象能做到仅为某一特定的HTTP请求提供数据(因为没有数据共享，所以对象是线程安全的)

web.ctx保存每个HTTP请求的特定信息，比如客户端环境变量。假设，我们想知道正在访问某页面的用户是从哪个网页跳转而来的：

例子

```
class example:
    def GET(self):
        referer = web.ctx.env.get('HTTP_REFERER', 'http://google.co
        raise web.seeother(referer)
```

上述代码用web.ctx.env获取HTTP_REFERER的值。如果HTTP__REFERER不存在，就会将google.com做为默认值。接下来，用户就会被重定向回到之前的来源页面。

web.ctx另一个特性，是它可以被loadhook赋值。例如：当一个请求被处理时，会话(Session)就会被设置并保存在web.ctx中。由于web.ctx是线程安全的，所以我们可以象使用普通的python对象一样，来操作会话(Session)。

'ctx'中的数据成员

Request

- `environ` 又被写做. `env` – 包含标准WSGI环境变量的字典。
- `home` – 应用的http根路径(译注：可以理解为应用的起始网址，协议＋站点域名＋应用所在路径)例：*http://example.org/admin*

- `homedomain` – 应用所在站点(可以理解为协议＋域名) *http://example.org*
- `homepath` – 当前应用所在的路径，例如： */admin*
- `host` – 主机名（域名）＋用户请求的端口（如果没有的话，就是默认的80端口），例如： *example.org*, *example.org:8080*
- `ip` – 用户的IP地址，例如： *xxx.xxx.xxx.xxx*
- `method` – 所用的HTTP方法，例如： *GET*
- `path` – 用户请求路径，它是基于当前应用的相对路径。在子应用中，匹配外部应用的那部分网址将被去掉。例如：主应用在 `code.py` 中，而子应用在 `admin.py` 中。在 `code.py` 中, 我们将 `/admin` 关联到 `admin.app` 。在 `admin.py` 中, 将 `/stories` 关联到 `stories` 类。在 `stories` 中, `web.ctx.path` 就是 `/stories` , 而非 `/admin/stories` 。形如： */articles/845*
- `protocol` – 所用协议，例如： *https*
- `query` – 跟在'？'字符后面的查询字符串。如果不存在查询参数，它就是一个空字符串。例如： *?fourlegs=good&twolegs=bad*
- `fullpath` 可以视为 `path + query` – 包含查询参数的请求路径，但不包括'homepath'。例如：*/articles/845?fourlegs=good&twolegs=bad*

Response

- `status` – HTTP状态码（默认是'200 OK') *401 Unauthorized* 未经授权
- `headers` – 包含HTTP头信息(headers)的二元组列表。
- `output` – 包含响应实体的字符串。

Application processors

问题

如何使用应用处理器，加载钩子(loadhooks)和卸载钩子(unloadhook)？

解法

web.py可以在处理请求之前或之后，通过添加处理器(processor)来完成某些操作。

```
def my_processor(handler):
    print 'before handling'
    result = handler()
    print 'after handling'
    return result

app.add_processor(my_processor)
```

可以用加载钩子(loadhook)和卸载钩子(unloadhook)的方式来完成同样的操作，它们分别在请求开始之前和结束之后工作。

```
def my_loadhook():
    print "my load hook"

def my_unloadhook():
    print "my unload hook"

app.add_processor(web.loadhook(my_loadhook))
app.add_processor(web.unloadhook(my_unloadhook))
```

你可以在钩子中使用和修改全局变量，比如：web.header()

```
def my_loadhook():
    web.header('Content-type', "text/html; charset=utf-8")

app.add_processor(web.loadhook(my_loadhook))
```

提示: 你也可以在钩子中使用 web.ctx 和 web.input() 。

```
def my_loadhook():
    input = web.input()
    print input
```

如何使用web.background

注意！！ web.backgrounder已转移到web.py 3.X实验版本中，不再是发行版中的一部分。你可以在这里下载，要把它与application.py放置在同一目录下才能正运行。

介绍

web.background和web.backgrounder都是python装饰器，它可以让某个函式在一个单独的background线程中运行，而主线程继续处理当前的HTTP请求，并在稍后报告background线程的状态(事实上，后台函式的标准输出(stdout)被返回给启动该线程的"backrounder")。 译注：我本来想将background thread翻译为后台线程，后来认为作者本意是想表达“被background修饰的函式所在的线程”，最后翻译采用“background线程”

这样，服务器就可以在处理其他http请求的同时，快速及时地响应当前客户端请求。同时，background线程继续执行需要长时间运行的函式。

例子

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from web import run, background, backgrounder
from datetime import datetime; now = datetime.now
from time import sleep

urls = (
    '/', 'index',
    )

class index:
    @backgrounder
    def GET(self):
        print "Started at %s" % now()
        print "hit f5 to refresh!"
        longrunning()

@background
def longrunning():
    for i in range(10):
        sleep(1)
        print "%s: %s" % (i, now())

if __name__ == '__main__':
    run(urls, globals())
```

在请求http://localhost:8080/时，将自动重定向到类似http://localhost:8080/?_t=3080772748的网址(t后面的数字就是background线程id)，接下来(在点击几次刷新之后)就会看到如下信息：

```
Started at 2008-06-14 15:50:26.764474
hit f5 to refresh!
0: 2008-06-14 15:50:27.763813
1: 2008-06-14 15:50:28.763861
2: 2008-06-14 15:50:29.763844
3: 2008-06-14 15:50:30.763853
4: 2008-06-14 15:50:31.764778
5: 2008-06-14 15:50:32.763852
6: 2008-06-14 15:50:33.764338
7: 2008-06-14 15:50:34.763925
8: 2008-06-14 15:50:35.763854
9: 2008-06-14 15:50:36.763789
```

提示

web.py在background.threaddb字典中保存线程信息。这就很容易检查线程的状态；

```
class threaddbviewer:
    def GET(self):
        for k, v in background.threaddb.items():
            print "%s - %s" % ( k, v )
```

web.py并不会主动去清空threaddb词典，这使得输出(如http://localhost:8080/?_t=3080772748)会一直执行，直到内存被用满。

通常是在backgrounder函式中做线程清理工作，是因为backgrounder可以获得线程id(通过web.input()得到"_t"的值，就是线程id)，从而根据线程id来回收资源。这是因为虽然background能知道自己何时结束，但它无法获得自己的线程id，所以background无法自己完成线程清理。

还要注意 How not to do thread local storage with Python 在python中如何避免多线程本地存储 - 线程ID有时会被重用(可能会引发错误)

在使用web.background时，还是那句话——“小心为上”

自定义NotFound消息

问题

如何定义NotFound消息和其他消息？

解法

```
import web

urls = (...)
app =  web.application(urls, globals())

def notfound():
    return web.notfound("Sorry, the page you were looking for was n

    # You can use template result like below, either is ok:
    #return web.notfound(render.notfound())
    #return web.notfound(str(render.notfound()))

app.notfound = notfound
```

要返回自定义的NotFound消息，这么做即可：

```
class example:
    def GET(self):
        raise web.notfound()
```

也可以用同样的方法自定义500错误消息：

```
def internalerror():
    return web.internalerror("Bad, bad server. No donut for you.")

app.internalerror = internalerror
```

如何流传输大文件

问题

如何流传输大文件？

解法

要流传输大文件，需要添加传输译码(Transfer-Encoding)区块头，这样才能一边下载一边显示。否则，浏览器将缓冲所有数据直到下载完毕才显示。

如果这样写：直接修改基础字符串(例中就是j)，然后用Yield返回——是没有效果的。如果要使用Yield,就要向对所有内容使用yield。因为这个函式此时是一个产生器。(注：请处请详看Yield文档，在此不做过多论述。)

例子

```
# Simple streaming server demonstration
# Uses time.sleep to emulate a large file read
import web
import time

urls = (
    "/",    "count_holder",
    "/(.*)",  "count_down",
    )
app = web.application(urls, globals())

class count_down:
    def GET(self,count):
        # These headers make it work in browsers
        web.header('Content-type','text/html')
        web.header('Transfer-Encoding','chunked')
        yield '<h2>Prepare for Launch!</h2>'
        j = '<li>Liftoff in %s...</li>'
        yield '<ul>'
        count = int(count)
        for i in range(count,0,-1):
            out = j % i
            time.sleep(1)
            yield out
        yield '</ul>'
        time.sleep(1)
        yield '<h1>Lift off</h1>'

class count_holder:
    def GET(self):
        web.header('Content-type','text/html')
        web.header('Transfer-Encoding','chunked')
        boxes = 4
        delay = 3
        countdown = 10
        for i in range(boxes):
            output = '<iframe src="/%d" width="200" height="500"><
            yield output
            time.sleep(delay)

if __name__ == "__main__":
    app.run()
```

管理自带webserver日志

问题

如何操作web.py自带的webserver的日志？

解法

我们可以用wsgilog来操作内置的webserver的日志，并做其为中间件加到应用中。

如下，写一个Log类继承wsgilog.WsgiLog，在*init*中把参数传给基类，如这个例子：

```
import sys, logging
from wsgilog import WsgiLog, LogIO
import config

class Log(WsgiLog):
    def __init__(self, application):
        WsgiLog.__init__(
            self,
            application,
            logformat = '%(message)s',
            tofile = True,
            file = config.log_file,
            interval = config.log_interval,
            backups = config.log_backups
            )
        sys.stdout = LogIO(self.logger, logging.INFO)
        sys.stderr = LogIO(self.logger, logging.ERROR)
```

接下来，当应用运行时，传递一个引用给上例中的Log类即可(假设上面代码是'mylog'模块的一部分，代码如下)：

```
from mylog import Log
application = web.application(urls, globals())
application.run(Log)
```

用cherrypy提供SSL支持

问题

如何用内置的cheerypy提供SSL支持？

解法

```
import web

from web.wsgiserver import CherryPyWSGIServer

CherryPyWSGIServer.ssl_certificate = "path/to/ssl_certificate"
CherryPyWSGIServer.ssl_private_key = "path/to/ssl_private_key"

urls = ("/.*", "hello")
app = web.application(urls, globals())

class hello:
    def GET(self):
        return 'Hello, world!'

if __name__ == "__main__":
    app.run()
```

实时语言切换

问题:

如何实现实时语言切换？

解法:

- 首先你必须阅读 模板语言中的i18n支持, 然后尝试下面的代码。

文件: code.py

```
import os
import sys
import gettext
import web

# File location directory.
rootdir = os.path.abspath(os.path.dirname(__file__))

# i18n directory.
localedir = rootdir + '/i18n'

# Object used to store all translations.
allTranslations = web.storage()

def get_translations(lang='en_US'):
    # Init translation.
    if allTranslations.has_key(lang):
        translation = allTranslations[lang]
    elif lang is None:
        translation = gettext.NullTranslations()
    else:
        try:
            translation = gettext.translation(
                    'messages',
                    localedir,
                    languages=[lang],
                    )
        except IOError:
            translation = gettext.NullTranslations()
    return translation

def load_translations(lang):
    """Return the translations for the locale."""
    lang = str(lang)
    translation  = allTranslations.get(lang)
```

```
    if translation is None:
        translation = get_translations(lang)
        allTranslations[lang] = translation

        # Delete unused translations.
        for lk in allTranslations.keys():
            if lk != lang:
                del allTranslations[lk]
    return translation

def custom_gettext(string):
    """Translate a given string to the language of the application.
    translation = load_translations(session.get('lang'))
    if translation is None:
        return unicode(string)
    return translation.ugettext(string)

urls = (
'/', 'index'
)

render = web.template.render('templates/',
        globals={
            '_': custom_gettext,
            }
        )

app = web.application(urls, globals())

# Init session.
session = web.session.Session(app,
        web.session.DiskStore('sessions'),
        initializer={
            'lang': 'en_US',
            }
        )

class index:
    def GET(self):
        i = web.input()
        lang = i.get('lang', 'en_US')

        # Debug.
        print >> sys.stderr, 'Language:', lang

        session['lang'] = lang
        return render.index()

if __name__ == "__main__": app.run()
```

模板文件: templates/index.html.

```
$_('Hello')
```

不要忘记生成必要的po&mo语言文件。参考: 模板语言中的i18n支持

现在运行code.py:

```
$ python code.py
http://0.0.0.0:8080/
```

然后用你喜欢的浏览器访问下面的地址，检查语言是否改变:

```
http://your_server:8080/
http://your_server:8080/?lang=en_US
http://your_server:8080/?lang=zh_CN
```

你必须:

- 确保语言文件(en_US、zh_CN等)可以动态改变。
- 确保custom_gettext()调用越省资源约好。

参考:

- 这里有使用app.app_processor()的 另一个方案。

Sessions and user state 会话和用户状态

Sessions

问题

如何在web.py中使用session

解法

注意！！！：*session*并不能在调试模式*(Debug mode)*下正常工作，这是因为*session*与调试模试下的重调用相冲突*(*有点类似*firefox*下著名的*Firebug*插件，使用*Firebug*插件分析网页时，会在火狐浏览器之外单独对该网页发起请求，所以相当于同时访问该网页两次*)*，下一节中我们会给出在调试模式下使用*session*的解决办法。

`web.session` 模块提供session支持。下面是一个简单的例子——统计有多少人正在使用session(session计数器)：

```
import web
web.config.debug = False
urls = (
    "/count", "count",
    "/reset", "reset"
)
app = web.application(urls, locals())
session = web.session.Session(app, web.session.DiskStore('sessions

class count:
    def GET(self):
        session.count += 1
        return str(session.count)

class reset:
    def GET(self):
        session.kill()
        return ""

if __name__ == "__main__":
    app.run()
```

web.py在处理请求之前，就加载session对象及其数据；在请求处理完之后，会检查session数据是否被改动。如果被改动，就交由session对象保存。

上例中的 `initializer` 参数决定了session初始化的值，它是个可选参数。

如果用数据库代替磁盘文件来存储session信息，只要用 `DBStore` 代替 `DiskStore` 即可。使用DBStore需要建立一个表，结构如下：

```
 create table sessions (
    session_id char(128) UNIQUE NOT NULL,
    atime timestamp NOT NULL default current_timestamp,
    data text
);
```

`DBStore` 被创建要传入两个参数： `db` 对象和session的表名。

```
db = web.database(dbn='postgres', db='mydatabase', user='myname', p
store = web.session.DBStore(db, 'sessions')
session = web.session.Session(app, store, initializer={'count': 0})
```

‘web.config‘中的 `sessions_parameters` 保存着session的相关设置， `sessions_parameters` 本身是一个字典，可以对其修改。默认设置如下：

```
web.config.session_parameters['cookie_name'] = 'webpy_session_id'
web.config.session_parameters['cookie_domain'] = None
web.config.session_parameters['timeout'] = 86400, #24 * 60 * 60, #
web.config.session_parameters['ignore_expiry'] = True
web.config.session_parameters['ignore_change_ip'] = True
web.config.session_parameters['secret_key'] = 'fLjUfxqXtfNoIldA0A0J
web.config.session_parameters['expired_message'] = 'Session expired
```

- cookie_name - 保存session id的Cookie的名称
- cookie_domain - 保存session id的Cookie的domain信息
- timeout - session的有效时间 ，以秒为单位
- ignore_expiry - 如果为True，session就永不过期
- ignore_change_ip - 如果为False，就表明只有在访问该session的IP与创建该session的IP完全一致时，session才被允许访问。
- secret_key - 密码种子，为session加密提供一个字符串种子
- expired_message - session过期时显示的提示信息。

在调试模式下使用session

问题

如何在调试模式下使用session?

解法

使用web.py自带的webserver提供web服务时，web.py就运行在调试模式下。当然最简单的办法就是禁用调试，只要令 `web.config.debug = False` 即可。

```
import web
web.config.debug = False

# rest of your code
```

如果非要用调试模式下使用session，可以用非主流的一些办法。哈哈

因为调试模式支持模块重载入(重载入，绝非重载。是reload,而非override)，所以reloader会载入主模块两次，因此，就会创建两个session对象。但我们只要把session存储在全局的数据容器中，就能避免二次创建session。

下面这个例子就是把session保存在 `web.config` 中：

```
import web
urls = ("/", "hello")

app = web.application(urls, globals())

if web.config.get('_session') is None:
    session = web.session.Session(app, web.session.DiskStore('sess
    web.config._session = session
else:
    session = web.config._session

class hello:
   def GET(self):
       print 'session', session
       session.count += 1
       return 'Hello, %s!' % session.count

if __name__ == "__main__":
   app.run()
```

在template中使用session

`问题`：我想在模板中使用session（比如：读取并显示session.username）

`解决`：

在应用程序中的代码:

```
render = web.template.render('templates', globals={'context': sess
```

在模板中的代码:

```
<span>You are logged in as <b>$context.username</b></span>
```

你可以真正的使用任何符合语法的python变量名，比如上面用的*context*。我更喜欢在应用中直接使用'session'。

如何操作Cookie

问题

如何设置和获取用户的Cookie?

解法

对web.py而言，设置/获取Cookie非常方便。

设置Cookies

概述

```
setcookie(name, value, expires="", domain=None, secure=False):
```

- *name* `(string)` - Cookie的名称，由浏览器保存并发送至服务器。
- *value* `(string)` -Cookie的值，与Cookie的名称相对应。
- *expires* `(int)` - Cookie的过期时间，这是个可选参数，它决定cookie有效时间是多久。以秒为单位。它必须是一个整数，而绝不能是字符串。
- *domain* `(string)` - Cookie的有效域－在该域内cookie才是有效的。一般情况下，要在某站点内可用，该参数值该写做站点的域（比如.webpy.org），而不是站主的主机名（比如wiki.webpy.org）
- *secure* `(bool)` - 如果为True，要求该Cookie只能通过HTTPS传输。.

示例

用 `web.setcookie()` 设置cookie,如下:

```
class CookieSet:
    def GET(self):
        i = web.input(age='25')
        web.setcookie('age', i.age, 3600)
        return "Age set in your cookie"
```

用 GET方式调用上面的类将设置一个名为age,默认值是25的cookie(实际上，默认值25是在web.input中赋予i.age的，从而间接赋予 cookie，而不是在setcookie函式中直接赋予cookie的)。这个cookie将在一小时后(即3600秒)过期。

`web.setcookie()` 的第三个参数－"expires"是一个可选参数，它用来设定cookie过期的时间。如果是负数，cookie将立刻过期。如果是正数，就表示cookie的有效时间是多久，以秒为单位。如果该参数为空，cookie就永不过期。

获得Cookies

概述

获取Cookie的值有很多方法，它们的区别就在于找不到cookie时如何处理。

方法**1**（如果找不到**cookie**，就返回**None**）：

```
web.cookies().get(cookieName)
    #cookieName is the name of the cookie submitted by the browser
```

方法**2**（如果找不到**cookie**，就抛出**AttributeError**异常）：

```
foo = web.cookies()
foo.cookieName
```

方法**3**（如果找不到**cookie**，可以设置默认值来避免抛出异常）：

```
foo = web.cookies(cookieName=defaultValue)
foo.cookieName   # return the value (which could be default)
    #cookieName is the name of the cookie submitted by the browser
```

示例：

用 `web.cookies()` 访问cookie. 如果已经用 `web.setcookie()` 设置了Cookie, 就可以象下面这样获得Cookie:

```
class CookieGet:
    def GET(self):
        c = web.cookies(age="25")
        return "Your age is: " + c.age
```

这个例子为cookie设置了默认值。这么做的原因是在访问时，若cookie不存在，web.cookies()就会抛出异常，如果事先设置了默认值就不会出现这种情况。

如果要确认cookie值是否存在，可以这样做：

```
class CookieGet:
    def GET(self):
        try:
             return "Your age is: " + web.cookies().age
        except:
             # Do whatever handling you need to, etc. here.
             return "Cookie does not exist."
```

或

```
class CookieGet:
    def GET(self):
        age=web.cookies().get('age')
        if age:
            return "Your age is: %s" % age
        else:
            return "Cookie does not exist."
```

用户认证

原作者没有写完，但是可以参照下一节，写得很详细

问题

如何完成一个用户认证系统？

解法

用户认证系统由这几个部分组成：用户添加，用户登录，用户注销以及验证用户是否已登录。用户认证系统一般都需要一个数据库。在这个例子中，我们要用到MD5和SQLite。

# #

```
import hashlib
import web

def POST(self):
    i = web.input()

    authdb = sqlite3.connect('users.db')
    pwdhash = hashlib.md5(i.password).hexdigest()
    check = authdb.execute('select * from users where username=? ar
    if check:
        session.loggedin = True
        session.username = i.username
        raise web.seeother('/results')
    else: return render.base("Those login details don't work.")
```

注意

这仅仅是个例子，可不要在真实的生产环境中应用哦。

在PostgreSQL下实现用户认证

问题

- 如何利用PostgreSQL数据库实现一个用户认证系统？

解法

- 用户认证系统有很多功能。在这个例子中，将展示如何在PostgreSQL数据库环境下一步一步完成一个用户认证系统

必需

- 因为要用到make模板和postgreSQL数据库，所以要: import web from web.contrib.template import render_mako import pg

第一步：创建数据库

首先，为创建一个用户表。虽然这个表结构非常简单，但对于大部分项目来说都足够用了。

```
CREATE TABLE example_users
(
  id serial NOT NULL,
  user character varying(80) NOT NULL,
  pass character varying(80) NOT NULL,
  email character varying(100) NOT NULL,
  privilege integer NOT NULL DEFAULT 0,
  CONSTRAINT utilisateur_pkey PRIMARY KEY (id)
)
```

第二步：确定网址

登录和注销对应两个网址：

- "Login" 对应登录页
- "Reset" 对应注销页

```
urls = (
    '/login', 'login',
    '/reset', 'reset',
     )
```

第三步：判断用户是否登录

要判断用户是否已登录，是非常简单的，只要有个变量记录用户登录的状态即可。在login/reset类中使用这段代码:

```
def logged():
    if session.login==1:
        return True
    else:
        return False
```

第四步：简单的权限管理

我把我的用户划为四类：管理员，用户，读者（已登录），访客（未登录）。根据example_users表中定义的不同权限，选择不同的模板路径。

```
def create_render(privilege):
    if logged():
        if privilege==0:
            render = render_mako(
                directories=['templates/reader'],
                input_encoding='utf-8',
                output_encoding='utf-8',
                )
        elif privilege==1:
            render = render_mako(
                directories=['templates/user'],
                input_encoding='utf-8',
                output_encoding='utf-8',
                )
        elif privilege==2:
            render = render_mako(
                directories=['templates/admin'],
                input_encoding='utf-8',
                output_encoding='utf-8',
                )
    else:
        render = render_mako(
            directories=['templates/communs'],
            input_encoding='utf-8',
            output_encoding='utf-8',
            )
    return render
```

第五：登录(Login)和注销(Reset)的python类

现在，让我们用个轻松的方法来解决：- 如果你已登录，就直接重定向到login_double.html模板文件 - 否则，还是到login.html。

```
class login:
    def GET(self):
        if logged():
            render = create_render(session.privilege)
            return "%s" % (
                render.login_double()               )
        else:
            render = create_render(session.privilege)
            return "%s" % (
                render.login()
                )
```

- 好了。现在写POST()方法。从.html文件中，我们得到表单提交的变量值(见login.html)，并根据变量值得到example_users表中对应的user数据
- 如果登录通过了，就重定向到login_ok.html。

- 如果没通过，就重定向到login_error.html。

```
def POST(self):
        user, passwd = web.input().user, web.input().passwd
        ident = db.query("select * from example_users where user =
        try:
            if passwd==ident[0][2]:
                session.login=1
                session.privilege=ident[0][4]
                render = create_render(session.privilege)
                return "%s" % (
                        render.login_ok()
                        )
            else:
                session.login=0
                session.privilege=0
                render = create_render(session.privilege)
                return "%s" % (
                    render.login_error()
                    )
        except:
            session.login=0
            session.privilege=0
            render = create_render(session.privilege)
            return "%s" % (
                render.login_error()
                )
```

对于reset方法，只要清除用户session，再重定向到logout.html模板页即可。

```
class reset:
    def GET(self):
        session.login=0
        session.kill()
        render = create_render(session.privilege)
        return "%s" % (
            render.logout()
            )
```

6th: 第六步：HTML模板帮助

嗯，我认为没有人想看这个，但我喜欢把所有的信息都提供出来。最重要的就是login.html。

```
<FORM action=/login method=POST>
    <table id="login">
        <tr>
            <td>User: </td>
            <td><input type=text name='user'></td>
        </tr>
        <tr>
            <td>Password: </td>
            <td><input type="password" name=passwd></td>
        </tr>
        <tr>
            <td></td>
            <td><input type=submit value=LOGIN></td>
        </tr>
    </table>
</form>
```

第七：问题或疑问？

- 邮件：您可以联想我，我的邮箱是guillaume(at)process-evolution(dot)fr
- IRC：#webpy on irc.freenode.net (pseudo: Ephedrax)
- 翻译：我是法国人，我的英文不好...你可以修改我的文档(译注：哈哈，谦虚啥，你那是没见过wrongway的山东英文...)

在子应用下使用session

提示

这个解决方案是来自web.py邮件列表。this

问题

如何在子应用中使用session？

解法

web.py默认session信息只能在主应用中共享，即便在其他模块中import Session都不行。在app.py（或main.py）可以这样初始化session：

```
session = web.session.Session(app, web.session.DiskStore('sessions
initializer = {'test': 'woot', 'foo':''})
```

.. 接下来创建一个被web.loadhook加载的处理器(processor)

```
def session_hook():
    web.ctx.session = session

app.add_processor(web.loadhook(session_hook))
```

.. 在子应用(假设是sub-app.py)中，可以这样操作session:

```
print web.ctx.session.test
web.ctx.session.foo = 'bar'
```

Utils 实用工具

发送邮件

问题

在web.py中，如何发送邮件？

解法

在web.py中使用 `web.sendmail()` 发送邮件.

```
web.sendmail('cookbook@webpy.org', 'user@example.com', 'subject',
```

如果在 `web.config` 中指定了邮件服务器，就会使用该服务器发送邮件，否则，就根据 `/usr/lib/sendmail` 中的设置发送邮件。

```
web.config.smtp_server = 'mail.mydomain.com'
```

如果要发送邮件给多个收件人，就给to_address赋值一个邮箱列表。

```
web.sendmail('cookbook@webpy.org', ['user1@example.com', 'user2@exa
```

`cc` 和 `bcc` 关键字参数是可选的，分别表示抄送和暗送接收人。这两个参数也可以是列表，表示抄送/暗送多人。

```
web.sendmail('cookbook@webpy.org', 'user@example.com', 'subject',
```

`headers` 参数是一个元组，表示附加标头信息(Addition headers)

```
web.sendmail('cookbook@webpy.org', 'user@example.com', 'subject',
        cc='user1@example.com', bcc='user2@example.com',
        headers=({'User-Agent': 'webpy.sendmail', 'X-Mailer': 'webp
        )
```

如何用Gmail发送邮件

问题

如何用Gmail发送邮件？

解法

安装和维护邮件服务器通常是沉闷乏味的。所以如果你有Gmail帐号，就可以使用Gmail做为SMTP服务器来发送邮件，我们唯一要做的就只是在 `web.config` 中指定Gmail的用户名和密码。

```
web.config.smtp_server = 'smtp.gmail.com'
web.config.smtp_port = 587
web.config.smtp_username = 'cookbook@gmail.com'
web.config.smtp_password = 'secret'
web.config.smtp_starttls = True
```

设置好之后，web.sendmail就能使用Gmail帐号来发送邮件了，用起来和其他邮件服务器没有区别。

```
web.sendmail('cookbook@gmail.com', 'user@example.com', 'subject',
```

可以在这里了解有关Gmail设置的更多信息 GMail: Configuring other mail clients

用soaplib实现webservice

问题

如何用soaplib实现webservice?

解法

Optio的soaplib通过用装饰器指定类型，从而直接编写SOAP web service。而且它也是到目前为止，唯一为web service提供WSDL文档的Python类库。

```
import web
from soaplib.wsgi_soap import SimpleWSGISoapApp
from soaplib.service import soapmethod
from soaplib.serializers import primitive as soap_types

urls = ("/hello", "HelloService",
        "/hello.wsdl", "HelloService",
        )
render = web.template.Template("$def with (var)\n$:var")

class SoapService(SimpleWSGISoapApp):
    """Class for webservice """

    #__tns__ = 'http://test.com'

    @soapmethod(soap_types.String,_returns=soap_types.String)
    def hello(self,message):
        """ Method for webservice"""
        return "Hello world "+message

class HelloService(SoapService):
    """Class for web.py """
    def start_response(self,status, headers):
        web.ctx.status = status
        for header, value in headers:
            web.header(header, value)

    def GET(self):
        response = super(SimpleWSGISoapApp, self).__call__(web.ctx
        return render("\n".join(response))

    def POST(self):
        response = super(SimpleWSGISoapApp, self).__call__(web.ctx
        return render("\n".join(response))

app=web.application(urls, globals())

if __name__ == "__main__":
    app.run()
```

可以用soaplib客户端测试一下：

```
>>> from soaplib.client import make_service_client
>>> from test import HelloService
>>> client = make_service_client('http://localhost:8080/hello', Hel
>>> client.hello('John')
'Hello world John'
```

可以在http://localhost:8080/hello.wsdl查看WSDL。

欲了解更多，请查看 soaplib,

Templates 模板

Templetor: web.py 模板系统

Introduction

web.py 的模板语言叫做 `Templetor` ，它能负责将 python 的强大功能传递给模板系统。 在模板中没有重新设计语法，它是类 python 的。 如果你会 python，你可以顺手拈来。

这是一个模板示例:

```
$def with (name)
Hello $name!
```

第一行表示模板定义了一个变量 `name` 。 第二行中的 `$name` 将会用 `name` 的值来替换。

如果是从 web.py 0.2 升级请看这里 升级 部分。

使用模板系统

通用渲染模板的方法：

```
render = web.template.render('templates')
return render.hello('world')
```

`render` 方法从模板根目录查找模板文件， `render.hello(..)` 表示渲染 hello.html 模板。实际上，系统会在根目录去查找叫 `hello` 的所有文件，直到找到匹配的。(事实上他只支持 .html 和 .xml 两种)

除了上面的使用方式，你也可以直接用文件的方式来处理模板 `frender` ：

```
hello = web.template.frender('templates/hello.html')
render hello('world')
```

直接使用字符串方式：

```
template = "$def with (name)\nHello $name"
hello = web.template.Template(template)
return hello('world')
```

语法

表达式用法

特殊字符 `$` 被用于特殊的 python 表达式。表达式能够被用于一些确定的组合当中 `()` 和 `{}`:

```
Look, a $string.
Hark, an ${arbitrary + expression}.
Gawk, a $dictionary[key].function('argument').
Cool, a $(limit)ing.
```

赋值

有时你可能需要定义一个新变量或给一些变量重新赋值，如下：

```
$ bug = get_bug(id)
<h1>$bug.title</h1>
<div>
    $bug.description
<div>
```

注意 `$` 在赋值变量名称之前要有一个空格，这有区别于常规的赋值用法。

过滤

模板默认会使用 `web.websafe` 过滤 html 内容(encodeing 处理)。

```
>>> render.hello("1 < 2")
"Hello 1 &lt; 2"
```

不需要过滤可以在 `$` 之后 使用 `:` 。示例：

```
该 Html 内容不会被义
$:form.render()
```

新起一行用法

在行末添加 `\` 代表显示层该内容不会被真实处理成一行。

```
If you put a backslash \
at the end of a line \
(like these) \
then there will be no newline.
```

转义 `$`

使用 `$$` 可以在输出的时候显示字符 `$`.

```
Can you lend me $$50?
```

注释

`$#` 是注释指示符。任何以 `$#` 开始的某行内容都被当做注释。

```
$# this is a comment
Hello $name.title()! $# display the name in title case
```

控制结构

模板系统支持 `for`, `while`, `if`, `elif` 和 `else`。像 python 一样，这里是需要缩进的。

```
$for i in range(10):
    I like $i

$for i in range(10): I like $i

$while a:
    hello $a.pop()

$if times > max:
    Stop! In the name of love.
$else:
    Keep on, you can do it.
```

`for` 循环内的成员变量只在循环内发生可用：

```
loop.index: the iteration of the loop (1-indexed)
loop.index0: the iteration of the loop (0-indexed)
loop.first: True if first iteration
loop.last: True if last iteration
loop.odd: True if an odd iteration
loop.even: True if an even iteration
loop.parity: "odd" or "even" depending on which is true
loop.parent: the loop above this in nested loops
```

有时候，他们使用起来很方便：

```
<table>
$for c in ["a", "b", "c", "d"]:
    <tr class="$loop.parity">
        <td>$loop.index</td>
        <td>$c</td>
    </tr>
</table>
```

其他

使用 `def`

可以使用 `$def` 定义一个新的模板函数，支持使用参数。

```
$def say_hello(name='world'):
    Hello $name!

$say_hello('web.py')
$say_hello()
```

其他示例：

```
$def tr(values):
    <tr>
    $for v in values:
        <td>$v</td>
    </tr>

$def table(rows):
    <table>
    $for row in rows:
        $:row
    </table>

$ data = [['a', 'b', 'c'], [1, 2, 3], [2, 4, 6], [3, 6, 9] ]
$:table([tr(d) for d in data])
```

代码

可以在 `code` 块书写任何 python 代码： $code: x = "you can write any python code here" y = x.title() z = len(x + y)

```
    def limit(s, width=10):
        """limits a string to the given width"""
        if len(s) >= width:
            return s[:width] + "..."
        else:
            return s

And we are back to template.
The variables defined in the code block can be used here.
For example, $limit(x)
```

使用 `var`

`var` 块可以用来定义模板结果的额外属性：

```
$def with (title, body)

$var title: $title
$var content_type: text/html

<div id="body">
$body
</div>
```

以上模板内容的输出结果如下：

```
>>> out = render.page('hello', 'hello world')
>>> out.title
u'hello'
>>> out.content_type
u'text/html'
>>> str(out)
'\n\n<div>\nhello world\n</div>\n'
```

内置 和 全局

像 python 的任何函数一样，模板系统同样可以使用内置以及局部参数。很多内置的公共方法像 `range` ， `min` ， `max` 等，以及布尔值 `True` 和 `False` ，在模板中都是可用的。部分内置和全局对象也可以使用在模板中。

全局对象可以使用参数方式传给模板，使用 `web.template.render` ：

```
import web
import markdown

globals = {'markdown': markdown.markdown}
render = web.template.render('templates', globals=globals)
```

内置方法是否可以在模板中也是可以被控制的：

```
# 禁用所有内置方法
render = web.template.render('templates', builtins={})
```

安全

模板的设计想法之一是允许非高级用户来写模板，如果要使模板更安全，可在模板中禁用以下方法：

- 不安全部分像 `import` ， `exec` 等；
- 允许属性开始部分使用 `_` ；
- 不安全的内置方法 `open` , `getattr` , `setattr` 等。

如果模板中使用以上提及的会引发异常 `SecurityException` 。

从 web.py 0.2 升级

新版本大部分兼容早期版本，但仍有部分使用方法会无法运行，看看以下原因：

- Template output is always storage like `TemplateResult` object, however

converting it to `unicode` or `str` gives the result as unicode/string.
- 重定义全局变量将无法正常运行，如果 x 是全局变量下面的写法是无法运行的。

```
$ x = x + 1
```

以下写法仍被支持，但不被推荐。

- 如果你原来用 `\$` 反转美元字符串， 推荐用 `$$` 替换；
- 如果你有时会修改 `web.template.Template.globals` ，建议通过向 `web.template.render` 传变量方式来替换。

站点布局模板

问题

如何让站点每个页面共享一个整站范围的模板？（在某些框架中，称为模板继承，比如ASP.NET中的母版页）

方法

我们可以用 base 属性来实现:

```
render = web.template.render('templates/', base='layout')
```

现在如果你调用 `render.foo()` 方法，将会加载 `templates/foo.html` 模板，并且它将会被 `templates/layout.html` 模板包裹。

"layout.html" 是一个简单模板格式文件，它包含了一个模板变量，如下:

```
$def with (content)
<html>
<head>
    <title>Foo</title>
</head>
<body>
$:content
</body>
</html>
```

在某些情况，如果不想使用基本模板，只需要创建一个没有base属性的reander对象，如下：

```
render_plain = web.template.render('templates/')
```

Tip: 在布局文件（layout.html）中定义的页面标题变量，如何在其他模板文件中赋值，如下:

templates/index.html

```
$var title: This is title.

<h3>Hello, world</h3>
```

templates/layout.html

```
$def with (content)
<html>
<head>
    <title>$content.title</title>
</head>
<body>
$:content
</body>
</html>
```

Tip: 在其他模板中引用css文件，如下:

templates/login.html

```
$var cssfiles: static/login.css static/login2.css

hello, world.
```

templates/layout.html

```
$def with (content)
<html>
<head>
    <title>$content.title</title>

    $if content.cssfiles:
        $for f in content.cssfiles.split():
            <link rel="stylesheet" href="$f" type="text/css" media=

</head>
<body>
$:content
</body>
</html>
```

输入的HTML代码如下:

```
<link rel="stylesheet" href="static/login.css" type="text/css" medi
<link rel="stylesheet" href="static/login2.css" type="text/css" med
```

交替风格

问题:

你想通过数据集合动态的生成交替背景色的列表.

方法:

Give templetor access to the `int` built-in and use modulo to test.

code.py

```
web.template.Template.globals['int'] = int
```

template.html

```
<ul>
$var i: 0
$for track in tracks:
    $var i: ${int(self.i) + 1}
    <li class="
    $if int(self.i) % 2:
        odd
    $else:
        even
    ">$track.title</li>
</ul>
```

New Templetor

In the new implementation of templetor (which will be the default when version .3 is released), within any template loop you have access to a $loop variable. This works like so:

```
<ul>
$for foo in foos:
    <li class="$loop.parity">
    $foo
    </li>
</ul>
```

Import functions into templates

`Problem` : How can I import a python module in template?

`Solution` :

While you write templates, inevitably you will need to write some functions which is related to display logic only. web.py gives you the flexibility to write large blocks of code, including defining functions, directly in the template using `$code` blocks (if you don't know what is $code block, please read the tutorial for Templator first). For example, the following code block will translate a status code from database to a human readable status message:

```
def status(c):
    st = {}
    st[0] = 'Not Started'
    st[1] = 'In Progress'
    st[2] = 'Finished'
    return st[c]
```

As you do more web.py development, you will write more such functions here and there in your templates. This makes the template messy and is a violation of the DRY (Don't Repeat Yourself) principle.

Naturally, you will want to write a module, say *displayLogic.py* and import that module into every templates that needs such functionalities. Unfortunately, `import` is disabled in template for security reason. However, it is easy to solve this problem, you can import any function via the global namespace into the template:

```
#in your application.py:
def status(c):
    st = {}
    st[0] = 'Not Started'
    st[1] = 'In Progress'
    st[2] = 'Finished'
    return st[c]

render = web.template.render('templates', globals={'stat':status})

#in the template:
$def with(status)
... ...
<div>Status: $stat(status)</div>
```

Remember that you can import more than one name into the *globals* dict. This trick is also used in importing session variable into template.

i18n support in template file

模板文件中的i18n支持

问题:

在web.py的模板文件中, 如何得到i18n的支持?

Solution:

项目目录结构:

```
proj/
   |- code.py
   |- i18n/
       |- messages.po
       |- en_US/
            |- LC_MESSAGES/
                   |- messages.po
                   |- messages.mo
   |- templates/
       |- hello.html
```

文件: proj/code.py

```
#!/usr/bin/env python
# encoding: utf-8

import web
import gettext

urls = (
    '/.*', 'hello',
    )

# File location directory.
curdir = os.path.abspath(os.path.dirname(__file__))

# i18n directory.
localedir = curdir + '/i18n'

gettext.install('messages', localedir, unicode=True)
gettext.translation('messages', localedir, languages=['en_US']).ins
render = web.template.render(curdir + '/templates/', globals={'_':

class hello:
    def GET(self):
        return render.hello()

# 使用内建的HTTP服务器来运行.
app = web.application(urls, globals())
if __name__ == "__main__":
    app.run()
```

模板文件: proj/templates/hello.html.

```
$_("Message")
```

创建一个locale目录并使用python2.6内建的pygettext.py从python脚本和模板文件中导出翻译:

```
shell> cd /path/to/proj/
shell> mkdir -p i18n/en_US/LC_MESSAGES/
shell> python /path/to/pygettext.py -a -v -d messages -o i18n/messa
Working on code.py
Working on templates/hello.html
```

你将会得到pot file: i18n/messages.po. 它的内容和下面的差不多 ('msgstr'包含了翻译后的信息):

```
 # 文件 code.py:40
msgid "Message"
msgstr "This is translated message in file: code.py."
```

拷贝文件'i18n/messages.po'到目录'i18n/en_US/LC_MESSAGES/'下, 然后翻译它. 使用gettext包的msgfmt工具或者使用python2.6内建的'msgfmt.py'文件将一个pot文件编译称mo文件:

```
shell> msgfmt -o i18n/en_US/LC_MESSAGES/messages.mo i18n/en_US/LC_M
```

运行web.py的服务器:

```
shell> cd /path/to/proj/
shell> python code.py
http://0.0.0.0:8000/
```

打开你的浏览器, 比如说firefox, 然后访问地址: http://192.168.0.3:8000/, 你将会看过翻译过的信息.

在webpy中使用Mako模板引擎

问题

如何在webpy中使用Mako模板引擎?

解决方案

首先需要安装Mako和web.py(0.3):http://www.makotemplates.org/ 然后尝试下面的代码:

```
# encoding: utf-8
# File: code.py
import web
from web.contrib.template import render_mako
urls = (
        '/(.*)', 'hello'
        )
app = web.application(urls, globals(), autoreload=True)
# input_encoding and output_encoding is important for unicode
# template file. Reference:
# http://www.makotemplates.org/docs/documentation.html#unicode
render = render_mako(
        directories=['templates'],
        input_encoding='utf-8',
        output_encoding='utf-8',
        )

class hello:
    def GET(self, name):
        return render.hello(name=name)
        # Another way:
        #return render.hello(**locals())

if __name__ == "__main__":
    app.run()
```

模板文件

```
## File: templates/hello.html

Hello, ${name}.
```

注意:

如果你使用Apache+mod_wsgi来部署webpy程序, 你也许会在Apache错误日志中得到下面的错误信息: [Sat Jun 21 21:56:22 2008] [error] [client 192.168.122.1] TopLevelLookupException: Cant locate template for uri 'index.html'

你必须使用绝对路径指出模板的位置. 你也可以使用相对路径来让它更简单一些:

```
import os

render = render_mako(
        directories=[os.path.join(os.path.dirname(__file__), 'templ
        input_encoding='utf-8',
        output_encoding='utf-8',
        )
```

参考:

http://code.google.com/p/modwsgi/wiki/ApplicationIssues

在webpy中使用Cheetah模板引擎

问题：

怎样在webpy中使用Cheetah模板引擎？

解决：

您需要先安装webpy(0.3)和Cheetah：http://www.cheetahtemplate.org/. 然后尝试使用下面的代码段：

```
# encoding: utf-8
# File: code.py

import web
from web.contrib.template import render_cheetah

render = render_cheetah('templates/')

urls = (
    '/(first)', 'first',
    '/(second)', 'second'
    )

app = web.application(urls, globals(), web.reloader)

class first:
    def GET(self, name):
        # cheetah template takes only keyword arguments,
        # you should call it as:
        #   return render.hello(name=name)
        # Below is incorrect:
        #   return render.hello(name)
        return render.first(name=name)

class second:
    def GET(self, name):
        return render.first(**locals())

if __name__ == "__main__":
    app.run()
```

模板文件

```
## File: templates/first.html

hello, $name.
```

Use Jinja2 template engine in webpy

问题

如何在web.py中使用Jinja2 (http://jinja.pocoo.org/2/) 模板引擎?

方案

首先需要安装Jinja2和webpy(0.3), 然后使用下面的代码做测试:

```
import web
from web.contrib.template import render_jinja

urls = (
        '/(.*)', 'hello'
        )

app = web.application(urls, globals())

render = render_jinja(
        'templates',   # 设置模板路径.
        encoding = 'utf-8', # 编码.
    )

#添加或者修改一些全局方法.
#render._lookup.globals.update(
#       var=newvar,
#       var2=newvar2,
#)

class hello:
    def GET(self, name):
        return render.hello(name=name)

if __name__ == "__main__":
    app.run()
```

模板文件: templates/hello.html

```
Hello, .
```

How to use templates on Google App Engine

问题

如何在 Google App Engine 上使用模板

解答

web.py templetor 把模板编译成 python 字节码，这需要访问标准库中的 parser 模块。不幸的是，由于安全原因 GAE 禁用了这个模块。

为了克服这个状况，web.py 支持把模板编译成 python 代码，从而避免在 GAE 上使用原来的模板。web.py 确保在应用这种方法的时候模板中的代码不需要任何改变。

为了编译一个文件夹中所有的模板（一旦有模板改动，就需要重新运行），运行：

```
$ python web/template.py --compile templates
```

以上命令把 templates/ 目录下的模板文件递归地全部编译，并且生产 `__init__.py` ，'web.template.render` 重新编写过，它将视 templates 为一个 python 模块。

Testing 测试

Testing with Paste and Nose

Problem

You want to test your web.py application.

Solution

```
from paste.fixture import TestApp
from nose.tools import *
from code import app

class TestCode():
    def test_index(self):
        middleware = []
        app = TestApp(app.wsgifunc(*middleware))
        r = app.get('/')
        assert_equal(r.status, 200)
        r.mustcontain('Hello, world!')
```

Background

This example makes use of the Paste and Nose libraries. Paste lets you throw test requests at your application, and adds some helpful custom methods to the response objects, such as mustcontain(), seen above. Nose makes writing and running your tests dead simple. When run from the base of your tree, it automatically finds and runs anything which is named like a test, adding necessary modules to your PYTHONPATH. This gives you the flexibility to run your tests from other directories, as well. Another benefit of Nose is that you no longer need to have every test class inherit from unittest.TestCase. Many more details are outlined on the project page.

Explanation

This code resides in a file called test_code.py. The directory layout of the application looks like this:

```
./
code.py
./test
    test_code.py
```

Most of the code example above should be fairly self-explanatory. From our main module, code, we import app, which is defined in the usual way:

```
app = web.application(urls, globals())
```

To set up the test, we pass its wsgifunc() to Paste's TestApp, as you have already seen in the example.

```
app = TestApp(app.wsgifunc(*middleware))
```

assert_equal() is one of the methods provided by nose's utils, and works just like unittest's assertEqual().

Setting Up the Test Environment

In order to avoid kicking off web.py's webserver when we run our tests, a change is required to the line which calls run(). It normally looks something like this:

```
if __name__ == "__main__": app.run()
```

We can define an environment variable, such as WEBPY_ENV=test, when we run our tests. In that case, the above line becomes the following:

```
import os

def is_test():
    if 'WEBPY_ENV' in os.environ:
        return os.environ['WEBPY_ENV'] == 'test'

if (not is_test()) and __name__ == "__main__": app.run()
```

Then, it's simply a matter of running nosetests like so:

```
WEBPY_ENV=test nosetests
```

The is_test() function comes in handy for other things, such as doing conditional database commits to avoid test database pollution.

RESTful doctesting using app.request

```
## !/usr/bin/env python

"""
RESTful web.py testing

usage: python webapp.py 8080 [--test]

>>> req = app.request('/mathematicians', method='POST')
>>> req.status
'400 Bad Request'

>>> name = {'first': 'Beno\xc3\xaet', 'last': 'Mandelbrot'}
>>> data = urllib.urlencode(name)
>>> req = app.request('/mathematicians', method='POST', data=data)
>>> req.status
'201 Created'
>>> created_path = req.headers['Location']
>>> created_path
'/mathematicians/b-mandelbrot'
>>> fn = '<h1 class=fn>{0} {1}</h1>'.format(name['first'], name['la
>>> assert fn in app.request(created_path).data

"""

import doctest
import urllib
import sys

import web

paths = (
  '/mathematicians(/)?', 'Mathematicians',
  '/mathematicians/([a-z])-([a-z]{2,})', 'Mathematician'
)
app = web.application(paths, globals())

dbname = {True: 'test', False: 'production'}[sys.argv[-1] == '--tes
db = {} # db = web.database(..., db='math_{0}'.format(dbname))

class Mathematicians:

  def GET(self, slash=False):
    """list all mathematicians and form to create new one"""
    if slash:
        raise web.seeother('/mathematicians')
    mathematicians = db.items() # db.select(...)
    return web.template.Template("""$def with (mathematicians)
```

```
        <!doctype html>
        <html>
        <head>
          <meta charset=utf-8>
          <title>Mathematicians</title>
        </head>
        <body>
          <h1>Mathematicians</h1>
          $if mathematicians:
            <ul class=blogroll>
              $for path, name in mathematicians:
                <li class=vcard><a class="fn url"
                href=/mathematicians/$path>$name.first $name.last</a>
            </ul>
          <form action=/mathematicians method=post>
            <label>First <input name=first type=text></label>
            <label>Last <input name=last type=text></label>
            <input type=submit value=Add>
          </form>
        </body>
        </html>""")(mathematicians)

    def POST(self, _):
      """create new mathematician"""
      name = web.input('first', 'last')
      key = '{0}-{1}'.format(name.first[0].lower(), name.last.lower()
      name.first, name.last = name.first.capitalize(), name.last.capi
      db[key] = name # db.insert(...)
      path = '/mathematicians/{0}'.format(key)
      web.ctx.status = '201 Created'
      web.header('Location', path)
      return web.template.Template("""$def with (path, name)
        <!doctype html>
        <html>
        <head>
          <meta charset=utf-8>
          <title>Profile Created</title>
        </head>
        <body>
          <p>Profile created for <a href=$path>$name.first $name.last
        </body>
        </html>""")(path, name)

  class Mathematician:

    def GET(self, first_initial, last_name):
      """display mathematician"""
      key = '{0}-{1}'.format(first_initial, last_name)
      try:
          mathematician = db[key] # db.select(...)
      except KeyError:
          raise web.notfound()
      return web.template.Template("""$def with (name)
```

```
        <!doctype html>
        <html>
        <head>
          <meta charset=utf-8>
          <title>$name.first $name.last</title>
        </head>
        <body class=vcard>
          <p><a href=/mathematicians rel=up>Mathematicians</a> &#x25B
          <h1 class=fn>$name.first $name.last</h1>
        </body>
        </html>""")(mathematician)

if __name__ == "__main__":
  if sys.argv[-1] == '--test':
    doctest.testmod()
  else:
    app.run()
```

User input 用户输入

File Upload Recipe

问题

如果你不是很了解表单上传或者CGI的话, 你会觉得文件上传有点奇特.

解决方法

```
import web

urls = ('/upload', 'Upload')

class Upload:
    def GET(self):
        return """<html><head></head><body>
<form method="POST" enctype="multipart/form-data" action="">
<input type="file" name="myfile" />
<br/>
<input type="submit" />
</form>
</body></html>"""

    def POST(self):
        x = web.input(myfile={})
        web.debug(x['myfile'].filename) # 这里是文件名
        web.debug(x['myfile'].value) # 这里是文件内容
        web.debug(x['myfile'].file.read()) # 或者使用一个文件对象
        raise web.seeother('/upload')

if __name__ == "__main__":
   app = web.application(urls, globals())
   app.run()
```

注意

需要注意以下内容:

- 表单需要一个enctype="multipart/form-data"的属性, 否则不会正常工作.
- 在webpy的代码里, 如果你需要默认值的话, myfile就需要默认值了(myfile={}), 文件会以字符串的形式传输 -- 这确实可以工作, 但是你会丢失文件的名称

保存上传的文件

问题

上传文件，并将其保存到预先设定的某个目录下。

方法

```
import web

urls = ('/upload', 'Upload')

class Upload:
    def GET(self):
        web.header("Content-Type","text/html; charset=utf-8")
        return """<html><head></head><body>
<form method="POST" enctype="multipart/form-data" action="">
<input type="file" name="myfile" />
<br/>
<input type="submit" />
</form>
</body></html>"""

    def POST(self):
        x = web.input(myfile={})
        filedir = '/path/where/you/want/to/save' # change this to t
        if 'myfile' in x: # to check if the file-object is created
            filepath=x.myfile.filename.replace('\\','/') # replaces
            filename=filepath.split('/')[-1] # splits the and choos
            fout = open(filedir +'/'+ filename,'w') # creates the f
            fout.write(x.myfile.file.read()) # writes the uploaded
            fout.close() # closes the file, upload complete.
        raise web.seeother('/upload')

if __name__ == "__main__":
   app = web.application(urls, globals())
   app.run()
```

Hang ups

同时还需要注意如下几点:

- 转到 fileupload。

- 千万不要让用户把文件上传到那些不经过文件后缀和类型检查而执行文件的文件夹下。
- 事实上，一定要以"mb"模式打开文件（在windows下），也就是二进制可写模式, 否则图片将无法上传。

上传文件大小限定

问题

如何限定上传文件的大小？

Solution

web.py 使用 `cgi` 模块来解析用户的输入， 而 `cgi` 模块对最大输入大小有限制。

下面的代码限制了最大数据输入为 10MB.

```
import cgi

# Maximum input we will accept when REQUEST_METHOD is POST
# 0 ==> unlimited input
cgi.maxlen = 10 * 1024 * 1024 # 10MB
```

请注意这是对POST方法提交数据大小的限制，而不是上传文件大小。当然如果表单中没有其他输入数据，上传文件完全可以达到限制的大小。

`cgi` 模块将会抛出 `ValueError` 异常，如果数据输入的大小超过了 `cgi.maxlen` 。我们可以捕捉该异常而避免显示不友好的错误信息。

```
class upload:
    def POST(self):
        try:
            i = web.input(file={})
        except ValueError:
            return "File too large"
```

web.input

web.input

问题

如何从form或是url参数接受用户数据.

解决方法

web.input()方法返回一个包含从url(GET方法)或http header(POST方法,即表单POST)获取的变量的web.storage对象(类似字典).举个例子,如果你访问页面 http://example.com/test?id=10,在Python后台你想取得 id=10 ,那么通过web.input()那就是小菜一碟:

```
class SomePage:
    def GET(self):
        user_data = web.input()
        return "<h1>" + user_data.id + "</h1>"
```

有时你想指定一个默认变量,而不想使用None.参考下面的代码:

```
class SomePage:
    def GET(self):
        user_data = web.input(id="no data")
        return "<h1>" + user_data.id + "</h1>"
```

注意,web.input()取得的值都会被当作string类型,即使你传递的是一些数字.

如果你想传递一个多值变量,比如像这样:

<select multiple="" size="3"><option>foo</option><option>bar</option><option>baz</option></select>

你需要让web.input知道这是一个多值变量,否则会变成一串而不是一个变量 .传递一个list给 web.input 作为默认值,就会正常工作.举个例子, 访问 http://example.com?id=10&id=20:

```
class SomePage:
    def GET(self):
        user_data = web.input(id=[])
        return "<h1>" + ",".join(user_data.id) + "</h1>"
```

译者补充: 多值变量这儿,在WEB上除了上面所说的multiple select 和query strings外,用得最多的就是复选框(checkbox)了,另外还有多文件上传时的<input type="file" ...>.

怎样使用表单 forms

问题：

怎样使用表单 forms

解决：

'web.form'模块提供支持创建，校验和显示表单。该模块包含一个'Form'类和各种输入框类如'Textbox'，'Password'等等。

当'form.validates()'调用时，可以针对每个输入检测的哪个是有效的，并取得校验理由列表。

'Form'类同样可以使用完整输入附加的关键字参数'validators'来校验表单。

这里是一个新用户注册的表单的示例：

```
import web
from web import form

render = web.template.render('templates') # your templates

vpass = form.regexp(r".{3,20}$", 'must be between 3 and 20 characte
vemail = form.regexp(r".*@.*", "must be a valid email address")

register_form = form.Form(
    form.Textbox("username", description="Username"),
    form.Textbox("email", vemail, description="E-Mail"),
    form.Password("password", vpass, description="Password"),
    form.Password("password2", description="Repeat password"),
    form.Button("submit", type="submit", description="Register"),
    validators = [
        form.Validator("Passwords did't match", lambda i: i.passwor

)

class register:
    def GET(self):
        # do $:f.render() in the template
        f = register_form()
        return render.register(f)

    def POST(self):
        f = register_form()
        if not f.validates():
            return render.register(f)
        else:
            # do whatever is required for registration
```

然后注册的模板应该像是这样：

```
$def with(form)

<h1>Register</h1>
<form method="POST">
    $:form.render()
</form>
```

个别显示表单字段

问题：

怎样在模板中个别显示表单字段？

解决：

你可以使用'render()'方法在你的模板中显示部分的表单字段。

假设你想创建一个名字/姓氏表单。很简单，只有两个字段，不需要验证，只是为了测试目的。

```
from web import form
simple_form = form.Form(
    form.Textbox('name', description='Name'),
    form.Textbox('surname', description='Surname'),
)
```

通常你可以使用 `simple_form.render（）` 或 `simple_form.render_css（）` 。但如你果你想一个一个的显示表单的字段，或者你怎样才能对模板中的表单显示拥有更多的控制权限？如果是这样，你可以对你的个别字段使用 `render()` 方法。

我们定义了两个字段名称为 `name` 和 `surname` 。这些名称将自动成为 `simple_form` 对象的属性。

```
>>> simple_form.name.render()
'<input type="text" name="name" id="name" />'
>>> simple_form.surname.render()
'<input type="text" name="surname" id="surname" />'
```

你同样可以通过类似的方法显示个别的描述：

```
>>> simple_form.surname.description
'Surname'
```

如果你有一个小模板片段（局部模板），你想统一的使用你所定义的所有表单字段？你可以使用表单对象的 `inputs` 属性迭代每个字段。下面是一个示例：

```
>>> for input in simple_form.inputs:
...     print input.description
...     print input.render()
...
Name
<input type="text" name="name" id="name" />
Surname
<input type="text" name="surname" id="surname" />
```

Database 数据库

多数据库使用

问题

如何在单独项目中应用多数据库?

解决办法

webpy 0.3 支持多数据库操作,并从web模块中移走数据库部分, 使其成为一个更典型的对象. 例子如下:

```
import web

db1 = web.database(dbn='mysql', db='dbname1', user='foo')
db2 = web.database(dbn='mysql', db='dbname2', user='foo')

print db1.select('foo', where='id=1')
print db2.select('bar', where='id=5')
```

增加, 更新, 删除和查询的方法跟原有单数据库操作类似.

当然, 你可以使用host和port参数来指定服务器地址和监听端口.

db.select 查询

问题:

怎样执行数据库查询?

解决方案:

如果是0.3版本, 连接部分大致如下:

```
db = web.database(dbn='postgres', db='mydata', user='dbuser', pw='
```

当获取数据库连接后, 可以这样执行查询数据库:

```
# Select all entries from table 'mytable'
entries = db.select('mytable')
```

select方法有下面几个参数:

- vars
- what
- where
- order
- group
- limit
- offset
- _test

vars

vars变量用来填充查询条件. 如:

```
myvar = dict(name="Bob")
results = db.select('mytable', myvar, where="name = $name")
```

what

what是标明需要查询的列名, 默认是*, 但是你可以标明需要查询哪些列.

```
results = db.select('mytable', what="id,name")
```

where

where查询条件, 如:

```
results = db.select('mytable', where="id>100")
```

order

排序方式:

```
results = db.select('mytable', order="post_date DESC")
```

group

按group组排列.

```
results = db.select('mytable', group="color")
```

limit

从多行中返回limit查询.

```
results = db.select('mytable', limit=10)
```

offset

偏移量, 从第几行开始.

```
results = db.select('mytable', offset=10)
```

_test

查看运行时执行的SQL语句:

```
results = db.select('mytable', offset=10, _test=True)
<sql: 'SELECT * FROM mytable OFFSET 10'>
```

db.upate 数据更新

问题

向数据库中更新数据。

解决方案

```
import web

db = web.database(dbn='postgres', db='mydata', user='dbuser', pw='
db.update('mytable', where="id = 10", value1 = "foo")
```

在 查询 中有更多关于可用参数的信息。

该更新操作会返回更新的影响行数。

db.delete 数据删除

问题

在数据库中删除数据。

解决办法

```
import web

db = web.database(dbn='postgres', db='mydata', user='dbuser', pw='
db.delete('mytable', where="id=10")
```

上面接受 "using" 和 "vars" 参数。

删除方法返回被删除的影响行数。

db.insert 向数据库中新增数据

问题

如何向数据加新增数据？

解决办法

在 0.3 中，数据库连接如下：

```
db = web.database(dbn='postgres', db='mydata', user='dbuser', pw='
```

数据库连接写好以后，“insert” 操作如下：

```
# 向 'mytable' 表中插入一条数据
sequence_id = db.insert('mytable', firstname="Bob",lastname="Smith'
```

上面的操作带入了几个参数，我们来说明一下：

- tablename
- seqname
- _test
- **values

tablename

表名，即你希望向哪个表新增数据。

seqname

可选参数，默认 None。Set `seqname` to the ID if it's not the default, or to `False` .

_test

`_test` 参数可以让你看到 SQL 的执行过程：

```
results = db.select('mytable', offset=10, _test=True)
><sql: 'SELECT * FROM mytable OFFSET 10'>
```

**values

字段参数。如果没有赋值，数据库可能创建默认值或者发出警告。

使用db.query进行高级数据库查询

问题：

您要执行的SQL语句如：高级的联接或计数。

解决：

webpy不会尝试为您和您的数据库建立层。相反，它试图以方便的通用任务，走出自己的方式，当您需要做的更高级的主题。执行高级的数据库查询是没有什么么不同。例如：

```
import web

db = web.database(dbn='postgres', db='mydata', user='dbuser', pw='

results = db.query("SELECT COUNT(*) AS total_users FROM users")
print results[0].total_users # -> prints number of entries in 'user
```

或者是，使用一个JOIN示例:

```
import web

db = web.database(dbn='postgres', db='mydata', user='dbuser', pw='

results = db.query("SELECT * FROM entries JOIN users WHERE entries
```

为了防止SQL注入攻击，db.query还接受了“vars”语法如下描述db.select:

```
results = db.query("SELECT * FROM users WHERE id=$id", vars={'id':1
```

这将避免用户输入，如果你信任这个“id”变量。

怎样使用数据库事务处理

问题：

怎样使用数据库事务处理？

解决：

数据库对象有一个方法“transaction”,将启动一个新的事务，并返回事务对象。这个事务对象可以使用commit提交事务或rollback来回滚事务。

```
import web

db = web.database(dbn="postgres", db="webpy", user="foo", pw="")
t = db.transaction()
try:
    db.insert('person', name='foo')
    db.insert('person', name='bar')
except:
    t.rollback()
    raise
else:
    t.commit()
```

在python 2.5+以上的版本，事务同样可以在段中使用：

```
from __future__ import with_statement

db = web.databse(dbn="postgres", db="webpy", user="foo", pw="")

with db.transaction():
    db.insert('person', name='foo')
    db.insert('person', name='bar')
```

它同样可能有一个嵌套的事务：

```
def post(title, body, tags):
    t = db.transaction()
    try:
        post_id = db.insert('post', title=title, body=body)
        add_tags(post_id, tags)
    except:
        t.rollback()
    else:
        t.commit()

def add_tags(post_id, tags):
    t = db.transaction()
    try:
        for tag in tags:
            db.insert('tag', post_id=post_id, tag=tag)
    except:
        t.rollback()
    else:
        t.commit()
```

嵌套的事务在sqlite中将被忽略，因为此特性不被sqlite支持。

sqlalchemy

问题

如何在web.py中使用sqlalchemy

方案

创建一个钩子并使用sqlalchemy的scoped session (http://www.sqlalchemy.org/docs/05/session.html#unitofwork_contextual)

```
import string
import random
import web

from sqlalchemy.orm import scoped_session, sessionmaker
from models import *

urls = (
    "/", "add",
    "/view", "view"
)

def load_sqla(handler):
    web.ctx.orm = scoped_session(sessionmaker(bind=engine))
    try:
        return handler()
    except web.HTTPError:
       web.ctx.orm.commit()
       raise
    except:
        web.ctx.orm.rollback()
        raise
    finally:
        web.ctx.orm.commit()

app = web.application(urls, locals())
app.add_processor(load_sqla)

class add:
    def GET(self):
        web.header('Content-type', 'text/html')
        fname = "".join(random.choice(string.letters) for i in rang
        lname = "".join(random.choice(string.letters) for i in rang
        u = User(name=fname
                ,fullname=fname + ' ' + lname
                ,password =542)
        web.ctx.orm.add(u)
        return "added:" + web.websafe(str(u)) \
                            + "<br/>" \
                            + '<a href="/view">view all</a>'

class view:
    def GET(self):
        web.header('Content-type', 'text/plain')
        return "\n".join(map(str, web.ctx.orm.query(User).all()))

if __name__ == "__main__":
    app.run()
```

models.py

```
from sqlalchemy import create_engine
from sqlalchemy import Column, Integer, String

engine = create_engine('sqlite:///mydatabase.db', echo=True)

from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()
class User(Base):
    __tablename__ = 'users'

    id = Column(Integer, primary_key=True)
    name = Column(String)
    fullname = Column(String)
    password = Column(String)

    def __init__(self, name, fullname, password):
        self.name = name
        self.fullname = fullname
        self.password = password

    def __repr__(self):
       return "<User('%s','%s', '%s')>" % (self.name, self.fullname

users_table = User.__table__
metadata = Base.metadata

if __name__ == "__main__":
    metadata.create_all(engine)
```

在跑程序之前,运行'python models.py'来初始化一次数据库.

整合 SQLite UDF (用户定义函数) 到 webpy 数据库层

问题：

用户在邮件列表中询问，我把它放在这里作为将来使用和参考。

解决：

您可以添加到Python函数到SQLite，并让它们在您的查询调用。

示例：

```
>>> import sqlite3 as db
>>> conn = db.connect(":memory:")
>>> conn.create_function("sign", 1, lambda val: val and (val > 0 ar
>>> cur = conn.cursor()
>>> cur.execute("select 1, -1")
<sqlite3.Cursor object at 0xb759f2c0>
>>> print cur.fetchall()
[(1, -1)]
>>> cur.execute("select sign(1), sign(-1), sign(0), sign(-99), sigr
<sqlite3.Cursor object at 0xb759f2c0>
>>> print cur.fetchall()
[(1, -1, 0, -1, 1)]
>>> conn.close()
```

在webpy中，你可以通过游标如db._db_cursor().connection 取得连接对象的引用。

示例：

```
>>> import web
>>> db = web.database(dbn="sqlite", db=":memory:")
>>> db._db_cursor().connection.create_function("sign", 1, lambda va
>>> print db.query("select sign(1), sign(-1), sign(0), sign(-99), s
[<Storage {'sign(1)': 1, 'sign(-1)': -1, 'sign(99)': 1, 'sign(-99)
```

使用字典动态构造where子句

问题

你希望创建一个字典来构造动态的where子句并且希望能够在查询语句中使用。

解决

```
>>> import web
>>> db = web.database(dbn='postgres', db='mydb', user='postgres')
>>> where_dict = {'col1': 1, col2: 'sometext'}
>>> db.delete('mytable', where=web.db.sqlwhere(where_dict), _test=T
<sql: "DELETE FROM mytable WHERE col1 = 1 AND col2 = 'sometext'">
```

解释

`web.db.sqlwhere` takes a Python dictionary as an argument and converts it into a string useful for where clause in different queries. You can also use an optional `grouping` argument to define the exact gouping of the individual keys. For instance:

`web.db.sqlwhere` 将Python的字典作为参数并且转换为适用于不同的查询语句的where子句的string类型数据。你也可以使用 `grouping` 参数来定义链接字典中的key的链接字符。例子如下。

```
>>> import web
>>> web.db.sqlwhere({'a': 1, 'b': 2}, grouping=' OR ')
'a = 1 OR b = 2'
```

`grouping` 的默认值为 `' AND '` .

Deployment 部署

通过Fastcgi和lighttpd部署

如果你对这个主题有任何问题，可以点击下面的链接访问相应的话题:

http://www.mail-archive.com/webpy@googlegroups.com/msg02800.html

下面的代码基于lighttpd 1.4.18，更高版本也可以工作

Note:

- 你可以重命名 `code.py` 为任何你自己愿意的名字，该例子还是以code.py为例。
- `/path-to/webpy-app` 为包含你的 `code.py` 代码的路径。
- `/path-to/webpy-app/code.py` 应该是你的**python file**的完整路径。

如果你还不确定你的lighttpd版本的话，你可以在命令行中使用 `lighttpd -v查看相应的版本信息。`

Note: 较早版本的lighttpd可能会按照不同的方式组织.conf文件，但是它们应该遵循的是相同的原则。

ligghttpd 在 Debian GNU/Linux 下的配置文件

```
Files and Directories in /etc/lighttpd:
---------------------------------------

lighttpd.conf:
         main configuration file

conf-available/
        This directory contains a series of .conf files. These file
        configuration directives necessary to load and run webserve
        If you want to create your own files they names should be
        build as nn-name.conf where "nn" is two digit number (numbe
        is used to find order for loading files)

conf-enabled/
        To actually enable a module for lighttpd, it is necessary t
        symlink in this directory to the .conf file in conf-availab

Enabling and disabling modules could be done by provided
/usr/sbin/lighty-enable-mod and /usr/sbin/lighty-disable-mod script
```

对于**web py**， 你需要允许 **mod_fastcgi** 模块和 **mod_rewrite**模块**,** 运行**:** **`/usr/sbin/lighty-enable-mod`** 启用 **`fastcgi`** （**Mac OS X**可能不需要） **(mod_rewrite** 模块可能需要启用 **`10-fastcgi.conf`** 文件**).**

下面是文件的基本结构（Mac OS X不同）：

- `/etc/lighttpd/lighttpd.conf`
- `/etc/lighttpd/conf-available/10-fastcgi.conf`
- `code.py`

对于Mac OS X或任何以Mac Ports邓方式安装的lighttpd，可以直接在路径下编写.conf文件并用lighttpd -f xxx.conf启动lighttpd，而无需去修改或考虑任何文件结构。

`/etc/lighttpd/lighttpd.conf`

```
server.modules              = (
            "mod_access",
            "mod_alias",
            "mod_accesslog",
            "mod_compress",
)
server.document-root       = "/path-to/webpy-app"
```

对我来说，我使用 postgresql，因此需要授予对的数据库权限，可以添加行如下（如果不使用则不需要）。

```
server.username = "postgres"
```

`/etc/lighttpd/conf-available/10-fastcgi.conf`

```
server.modules   += ( "mod_fastcgi" )
server.modules   += ( "mod_rewrite" )

 fastcgi.server = ( "/code.py" =>
 (( "socket" => "/tmp/fastcgi.socket",
    "bin-path" => "/path-to/webpy-app/code.py",
    "max-procs" => 1,
   "bin-environment" => (
     "REAL_SCRIPT_NAME" => ""
   ),
   "check-local" => "disable"
 ))
 )

如果本地的lighttpd跑不起来的话，需要设置check-local属性为disable。

 url.rewrite-once = (
   "^/favicon.ico$" => "/static/favicon.ico",
   "^/static/(.*)$" => "/static/$1",
   "^/(.*)$" => "/code.py/$1",
 )
```

`/code.py` 在代码头部添加以下代码，让系统环境使用系统环境中当前的python

```
#!/usr/bin/env python
```

最后不要忘记了要对需要执行的py代码设置执行权限，否则你可能会遇到“permission denied”错误。

```
$ chmod 755 /path-to/webpy-app/code.py
```

Webpy + Nginx with FastCGI搭建Web.py

这一节讲解的是如何使用Nginx和FastCGI搭建Web.py应用

环境依赖的软件包

- Nginx 0.8. *or 0.7.* (需要包含fastcgi和rewrite模块)。
- Webpy 0.32
- Spawn-fcgi 1.6.2
- Flup

注意：Flup是最常见的忘记装的软件，需要安装

更老的版本应该也可以工作，但是没有测试过，最新的是可以工作的

一些资源

- Nginx wiki
- Spawn-fcgi
- Flup

Notes

- 你可以重命名 `index.py` 为任何你想要的文件名。
- `/path/to/www` 为代码路径。
- `/path/to/www/index.py` 为python代码的完整路径。

Nginx 配置文件

```
location / {
    include fastcgi_params;
    fastcgi_param SCRIPT_FILENAME $fastcgi_script_name;  # [1]
    fastcgi_param PATH_INFO $fastcgi_script_name;        # [2]
    fastcgi_pass 127.0.0.1:9002;
}
```

对于静态文件可以添加如下配置:

```
location /static/ {
    if (-f $request_filename) {
    rewrite ^/static/(.*)$  /static/$1 break;
    }
}
```

注意: 地址和端口号可能会是不同的。

Spawn-fcgi

可以通过一下命令启动一个Spawn-fcgi进程:

```
spawn-fcgi -d /path/to/www -f /path/to/www/index.py -a 127.0.0.1 -p
```

启动和关闭的命令

启动:

```
#!/bin/sh
spawn-fcgi -d /path/to/www -f /path/to/www/index.py -a 127.0.0.1 -p
```

关闭:

```
#!/bin/sh
kill `pgrep -f "python /path/to/www/index.py"`
```

Note: 你可以随意填写地址和端口信息，但是一定需要和Nginx配置文件相匹配。

Hello world!

讲下面的代码保存为index.py（或者任何你喜欢的），注意，使用Nginx配置的话， `web.wsgi.runwsgi = lambda func, addr=None: web.wsgi.runfcgi(func` 这一行代码是必须的。

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import web

urls = ("/.*", "hello")
app = web.application(urls, globals())

class hello:
    def GET(self):
        return 'Hello, world!'

if __name__ == "__main__":
    web.wsgi.runwsgi = lambda func, addr=None: web.wsgi.runfcgi(fur
    app.run()
```

注意: 同样需要给代码设置权限，代码如下chmod +x index.py。

运行

1. 打开一个 `spawn-fcgi` 进程.
2. 打开 Nginx.

如果需要检查应用程序是否运行，使用 `ps aux|grep index.py` 可以很容易的查看。

重启nginx配置:

```
/path/to/nginx/sbin/nginx -s reload
```

停止nginx:

```
/path/to/nginx/sbin/nginx -s stop
```

注意：运行后可访问http://localhost访问网站，更多信息可以去参考nginx官方文档。

CGI deployment on Apache

Here are the simple steps needed to create and run an web.py application.

- Install web.py and flups
- Create the application as documented

```
if __name__ == "__main__":
     web.run(urls, globals())
```

For our example, let it be named `app.py`, located in `/www/app` and we need it accessible as `http://server/app/app.py`.

- Configure Apache (version 2.2 in this example)

```
ScriptAlias /app "/www/app/"
 &lt;Directory "/www/app/"&gt;
         Options +ExecCGI +FollowSymLinks
         Order allow,deny
         Allow from all
 &lt;/Directory&gt;
```

That's it. Your application is accessible via `http://server/app/app.py/`. Additional URLs handled by the application are added to the end of the URL, for examples `http://server/app/app.py/myurl`.

- .htaccess configuration

```
Options +ExecCGI
       AddHandler cgi-script .py
       DirectoryIndex index.py
       &lt;IfModule mod_rewrite.c&gt;
           RewriteEngine on
           RewriteBase /
           RewriteCond %{REQUEST_FILENAME} !-f
           RewriteCond %{REQUEST_FILENAME} !-d
           RewriteCond %{REQUEST_URI} !^/favicon.ico$
           RewriteCond %{REQUEST_URI} !^(/.*)+index.py/
           RewriteRule ^(.*)$ index.py/$1 [PT]
       &lt;/IfModule&gt;
```

Here it is assumed that your application is called index.py. The above htaccess checks if some static file/directory exists failing which it routes the data to your index.py. Change the Rewrite Base to a sub-directory if needed.

使用Apache + mod_wsgi部署webpy应用

下面的步骤在Apache-2.2.3 (Red Hat Enterprise Linux 5.2, x86_64),mod_wsgi-2.0中测试通过。（译者注：本人在Windows2003 + Apache-2.2.15 + mod_wsgi-3.0也测试通过）

注意：

- 您可以使用您自己的项目名称替换'appname'。
- 您可以使用您自己的文件名称替换'code.py'。
- /var/www/webpy-app 为包含您的code.py的文件夹目录路径。
- /var/www/webpy-app/code.py 是您的python文件的完整路径。

步骤：

- 下载和安装mod_wsgi从它的网站：

http://code.google.com/p/modwsgi/. 它将安装一个'.so'的模块到您的apache 模块文件夹，例如：

```
/usr/lib64/httpd/modules/
```

- 在httpd.conf中配置Apache 加载 mod_wsgi模块和您的项目：

```
LoadModule wsgi_module modules/mod_wsgi.so

 WSGIScriptAlias /appname /var/www/webpy-app/code.py/

 Alias /appname/static /var/www/webpy-app/static/
 AddType text/html .py

 &lt;Directory /var/www/webpy-app/&gt;
     Order deny,allow
     Allow from all
 &lt;/Directory&gt;
```

- 演示文件 'code.py':

```
import web

 urls = (
     '/.*', 'hello',
     )

 class hello:
     def GET(self):
         return "Hello, world."

 application = web.application(urls, globals()).wsgifunc()
```

- 在您的浏览器地址栏中输入' http://your_server_name/appname' 来验证它是否可用。

注意: mod_wsgi + sessions

如果您需要在mod_wsgi中使用sessions，您可以改变您的代码如下：

```
app = web.application(urls, globals())

curdir = os.path.dirname(__file__)
session = web.session.Session(app, web.session.DiskStore(curdir +

application = app.wsgifunc()
```

mod_wsgi 性能:

有关mod_wsgi的性能，请参考mod_wsgi的维基页：
http://code.google.com/p/modwsgi/wiki/PerformanceEstimates

deploying web.py with nginx and mod_wsgi

It is possible to deploy web.py with nginx using a mod_wsgi similar to the module for Apache.

After compiling and installing nginx with mod_wsgi, you can easily get a web.py app up and running with the following config* (edit the paths and settings with your own):

```
wsgi_python_executable  /usr/bin/python;

server {
    listen 80;
    server_name www.domain_name.com domain_name.com;
    root /path/to/your/webpy;

    include /etc/nginx/wsgi_vars;
    location / {
        wsgi_pass /path/to/your/webpy/app.py;
     }
}
```

*Note: This is a snippet of the relevant information to setup mod_wsgi for your web app and NOT a full config for running nginx.

Helpful links: nginx website wiki page on mod_wsgi

Webpy + Nginx with FastCGI搭建Web.py

这一节讲解的是如何使用Nginx和FastCGI搭建Web.py应用

环境依赖的软件包

- Nginx 0.8. *or 0.7.* (需要包含fastcgi和rewrite模块)。
- Webpy 0.32
- Spawn-fcgi 1.6.2
- Flup

注意：Flup是最常见的忘记装的软件，需要安装

更老的版本应该也可以工作，但是没有测试过，最新的是可以工作的

一些资源

- Nginx wiki
- Spawn-fcgi
- Flup

Notes

- 你可以重命名 `index.py` 为任何你想要的文件名。
- `/path/to/www` 为代码路径。
- `/path/to/www/index.py` 为python代码的完整路径。

Nginx 配置文件

```
location / {
    include fastcgi_params;
    fastcgi_param SCRIPT_FILENAME $fastcgi_script_name;  # [1]
    fastcgi_param PATH_INFO $fastcgi_script_name;        # [2]
    fastcgi_pass 127.0.0.1:9002;
}
```

对于静态文件可以添加如下配置:

```
location /static/ {
    if (-f $request_filename) {
    rewrite ^/static/(.*)$  /static/$1 break;
    }
}
```

注意: 地址和端口号可能会是不同的。

Spawn-fcgi

可以通过一下命令启动一个Spawn-fcgi进程:

```
spawn-fcgi -d /path/to/www -f /path/to/www/index.py -a 127.0.0.1 -p
```

启动和关闭的命令

启动:

```
#!/bin/sh
spawn-fcgi -d /path/to/www -f /path/to/www/index.py -a 127.0.0.1 -p
```

关闭:

```
#!/bin/sh
kill `pgrep -f "python /path/to/www/index.py"`
```

Note: 你可以随意填写地址和端口信息，但是一定需要和Nginx配置文件相匹配。

Hello world!

讲下面的代码保存为index.py（或者任何你喜欢的），注意，使用Nginx配置的话， `web.wsgi.runwsgi = lambda func, addr=None: web.wsgi.runfcgi(func` 这一行代码是必须的。

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import web

urls = ("/.*", "hello")
app = web.application(urls, globals())

class hello:
    def GET(self):
        return 'Hello, world!'

if __name__ == "__main__":
    web.wsgi.runwsgi = lambda func, addr=None: web.wsgi.runfcgi(fun
    app.run()
```

注意: 同样需要给代码设置权限，代码如下chmod +x index.py。

运行

1. 打开一个 `spawn-fcgi` 进程.
2. 打开 Nginx.

如果需要检查应用程序是否运行，使用 `ps aux|grep index.py` 可以很容易的查看。

重启nginx配置:

```
/path/to/nginx/sbin/nginx -s reload
```

停止nginx:

```
/path/to/nginx/sbin/nginx -s stop
```

注意：运行后可访问http://localhost访问网站，更多信息可以去参考nginx官方文档。